ALGORITMOS E PROGRAMAÇÃO

COM RESUMO DAS LINGUAGENS PASCAL E C

PROF. AFONSO INÁCIO ORTH

SUMÁRIO

	Introdução	3
Capítulo I	Conceitos Básicos de Algoritmos	4
	Exercícios	10
Capítulo II	Algoritmos Seqüenciais	11
	Questionário	16
	Exercícios	16
Capítulo III	Algoritmos com Seleção	18
	Questionário	25
	Exercícios	25
Capítulo IV	Algoritmos com Repetição	27
	Questionário	34
	Exercícios	34
Capítulo V	Algoritmos de Manipulação de Matrizes	40
	Exercícios	49
Capítulo VI	Subalgoritmos	55
	Subalgoritmos do Tipo Função	56
	Subalgoritmos do Tipo Procedimento	61
	Técnica Top-Down	67
	Técnica Bottom-up	70
	Questionário	73
	Exercícios	73
Capítulo VII	Algoritmos Recursivos	77
	Exercícios	83
Capítulo VIII	Algoritmos de Manipulação de Caracteres	84
	Implementação de Cadeias de Caracteres usando Vetores	86
	Exercícios	87
Capítulo IX	Registros	88
	Registro de Vetores	89
	Vetor de Registros	90
	Exercícios	96
Capítulo X	Correção e outras qualidades de um algoritmo	100
	Formas de melhoria da eficiência de um algoritmo	101
Apêndice A	Tradução de Algoritmos para PASCAL	105
Apêndice B	Tradução de Algoritmos para C	113
Bibliografia		122

INTRODUÇÃO

O uso de algoritmos é quase tão antigo quanto a matemática. Com o passar do tempo, entretanto, ele foi bastante esquecido pela matemática. Com o advento das máquinas de calcular e , mais tarde dos computadores, o uso de algoritmos ressurgiu com grande vigor, como uma forma de indicar o caminho para a solução dos mais variados problemas.

Segundo Knuth (68), o termo algoritmo é derivado do nome de um matemático persa do século IX, chamado Abu Ja´far Maomé ibn Mûsâ al-Khowârizm (1825).

Algoritmo não é a solução do problema, pois, se assim fosse, cada problema teria um único algoritmo. Algoritmo é um caminho para a solução de um problema, e, em geral, os caminhos que levam à uma solução são muitos. A solução é obtida por meio da execução do algoritmo, seja mentalmente, ou manualmente usando lápis e papel ou por meio de um computador

Ao longo dos anos surgiram muitas formas de representação dos algoritmos, algumas utilizando linguagens semelhantes às linguagens de programação, ou as próprias linguagens de programação e outras utilizando formas gráficas de representação dos algoritmos.

Dentre as formas de representação, nos últimos anos, deu-se acentuada preferência por formas estruturadas, cuja principal vantagem é a de facilitar a legibilidade e compreensão dos algoritmos.

Uma das dificuldades constatadas no aprendizado de algoritmos é a passagem de uma língua natural, de expressão completamente livre, para uma restrição da linguagem, rigidamente formalizada, com sintaxe não familiar e, em geral, em uma língua estrangeira.

Certamente, uma passagem por uma linguagem algorítmica, embora mais restrita que a língua natural, mas com sintaxe conhecida e em português, representa uma transição bem menos traumatizante.

Outro argumento razoável para justificar o uso de uma linguagem algorítmica em lugar de uma linguagem de programação ou até de uma linguagem algorítmica executável, é o fato de que, sob a óptica do ensino, deve-se ensinar um conceito cada vez, sob pena de se perder muito mais tempo para o aprendizado. O objetivo aqui é o de ensinar uma forma de expressar algoritmos e não uma sintaxe de uma linguagem de programação. Normalmente detalhes de notação como “;”, “:=”, etc. desviam a atenção do aluno do principal.

Perdemos muito tempo ensinando detalhes da sintaxe das linguagens de programação e, como este ensino se dá geralmente em momentos nos quais o aluno não está necessitando imediatamente destas informações e provavelmente não está motivado para aprendê-las, perdemos literalmente o nosso precioso tempo.

Além disso, o uso de uma linguagem algorítmica evita que os alunos associem a lógica da solução do problema com eventuais detalhes de implementação do algoritmo em tal ou qual linguagem.

A experiência tem mostrado que, uma vez dominando a linguagem algorítmica, o ensino de uma linguagem de programação pode ser feito em dois períodos de aula com folga, incluindo exercícios. Melhor ainda, deixemos que o aluno aprenda os detalhes das linguagens de programação no momento em que necessitar deles e estiver motivado para tanto. Não subestimemos a capacidade do aluno de buscar as informações que necessita. Isto é o cerne do processo ensino-aprendizagem, onde o processo **ensino** é muito menos importante do que o processo **aprendizagem**. **É o aluno que aprende quando motivado para tanto e não o professor que ensina.** Um aluno motivado aprende, inclusive, apesar do professor e de outras condições menos favoráveis.

Os algoritmos iniciais são, via de regra, suficientemente simples para se fazer uma verificação de seu funcionamento de forma manual. Com isso o aluno aprenderá mais facilmente o efetivo funcionamento de cada estrutura de controle da linguagem utilizada e, além disso, a verificação de algoritmos, pelo autor, é uma forma adicional e complementar de aprendizagem de algoritmos.

O aprendizado de algoritmos só se consegue através de muitos exercícios.

Algoritmos **NÃO** se aprendem:

- Copiando algoritmos;
- Estudando algoritmos.

Algoritmos **SÓ** se aprendem:

- Construindo algoritmos;
- Testando algoritmos.

CAPÍTULO I

CONCEITOS BÁSICOS DE ALGORÍTMOS

Este capítulo visa conceituar: constante, variável, operação, expressão e atribuição.
Considerando-se a fórmula matemática simples do cálculo do volume de uma esfera têm-se:

$$\mathbf{V} = \tfrac{4}{3}\boldsymbol{\pi}\mathbf{R}^3$$

onde encontramos:

1 – valores que podem ser classificados como:

a) valores constantes, invariantes em todas as aplicações da fórmula, no caso, os valores 4, 3, e π, que, pelo fato de serem sempre os mesmos são denominados de **constantes;**
b) valores a serem substituídos na fórmula, em cada aplicação que dela fazemos. A representação destes valores, usualmente é feita através do uso de letras ou combinações de letras e/ou dígitos numéricos, que recebem o nome de **variáveis** e tornam a fórmula genérica, isto é, possível de ser usada para resolver uma certa classe de problemas e não apenas um problema específico. No exemplo, **V** e **R** são denominados de variáveis. Outros exemplos de variáveis: a, b1, valor, x14, codigo10, etc.

2 - operações a serem executadas sobre determinados operandos (valores), para a obtenção da solução do problema. Na fórmula do volume da esfera estão representadas as operações de divisão, multiplicação e potenciação, efetuadas sobre os operandos 4, 3, π e R, numa seqüência pré-determinada, mas nem sempre rigorosa, devido às leis matemáticas da comutatividade e associatividade.

Uma fórmula matemática é simplesmente uma descrição de um conjunto de **ações** (operações) que devem ser executadas sobre um conjunto de **objetos** (constantes e/ou variáveis).

É importante que os objetos sobre os quais as operações tem efeito sejam **precisamente definidos**.

Existe uma outra relação que deve ser cuidada entre objetos e operações. Certas operações somente podem ser executadas sobre determinados objetos, se estes objetos satisfizerem certas condições como pode ser ilustrado nos exemplos abaixo:

Exemplo 1: $\sqrt{a}$

A operação de radiciação somente pode ser executada sobre o objeto **a**, se este pertencer ao conjunto dos reais positivos $\mathbf{R}^+$.

Exemplo 2: **n !**

O conjunto de operações necessárias para o cálculo do fatorial de **n** só fará sentido prático se o objeto **n** pertencer ao conjunto dos inteiros positivos $\mathbf{I}^+$.

Exemplo 3: **Arc sen x**

A operação de determinação do arco do seno trigonométrico somente pode ser executada sobre o objeto **x** se este pertencer ao intervalo **[-1,1]** do conjunto **R.**

Exemplo 4: Verificar a ocorrência da letra **a** em um dado texto.

A operação de comparação de cada um dos caracteres do texto com a letra **a** tem como resultado um valor lógico **verdadeiro** ou **falso** conforme o caracter **a** for ou não encontrado no texto. Entretanto, ela só fará sentido se o caracter **a** pertencer ao conjunto de símbolos possíveis de ocorrerem no texto.
Nos exemplos apresentados, pode-se notar dois tipos de limitações:

1. Limitações de classes de valores que podem ser:

Inteiros – quando pertencem aos conjuntos **I** ou **Z.**
Reais – quando pertencem ao conjunto **R .**
Lógicos – quando pertencem ao conjunto **{ verdadeiro, falso }**.
Literais – quando pertencem ao **conjunto dos caracteres.**

2. Limitações de intervalo, pois, algumas operações não atuam sobre todo o conjunto, mas somente sobre parte dele.

O conjunto de valores que uma variável pode assumir poderá influir, inclusive, na maneira de se executar certas operações:

Exemplo: $\mathbf{A^B}$ que pode assumir as seguintes variações de cálculo:

a) B **inteiro e positivo** : $A^{\mathbf{B}} = \mathbf{A \times A \times A \times \ldots A}$

b) B **inteiro e negativo** : $A^{\mathbf{B}} = \frac{1}{A} \times \frac{1}{A} \times \frac{1}{A} \times \ldots \times \frac{1}{A}$

c) B **real** : $A^{\mathbf{B}} = 10^{\mathbf{B.\ log_{10} A}}$

d) B **nulo** : $A^{\mathbf{B}} = \mathbf{1}$

Uma **constante** é pois, um objeto invariante em cada execução de uma fórmula matemática, enquanto **variável** é um objeto que representa um valor que pode ser alterado a cada execução da fórmula que a contém.

Se for resolvida a expressão $\mathbf{V} = \frac{4}{3}\boldsymbol{\pi}\mathbf{R^3}$ para uma esfera de raio 3, por exemplo, teremos:

1. Substituir R por 3 ($\mathbf{V} = \frac{4}{3}\boldsymbol{\pi}\,\mathbf{3^3}$)
2. Substituir π por 3,1416 ($\mathbf{V} = \frac{4}{3}\,\mathbf{3{,}1416.\ 3^3}$)
3. Calcular a potenciação ($\mathbf{V} = \frac{4}{3}\,\mathbf{3{,}1416\ .\ 27}$)
4. Calcular a divisão 4 por 3 (**V = 1,333. 3,1416. 27**)
5. Calcular a primeira multiplicação (**V = 4,1885 . 27**)
6. Calcular a multiplicação restante (**V = 113,0895**)

Observa-se que neste caso, a existência de números como $\frac{4}{3}$ **= 1,3333…** obrigou-nos ao estabelecimento de um limite de precisão para que a seqüência de operações pudesse ser executada.

As variáveis podem ser visualizadas como receptáculos de valores. Cada receptáculo é identificado através de um **nome** e pode receber um **valor** de um determinado tipo.

Assim a fórmula $\mathbf{V} = \frac{4}{3}\boldsymbol{\pi}\mathbf{R^3}$ pode ser visualizada como segue:

$$\mathbf{V} = \frac{4}{3}\boldsymbol{\pi}\ \mathbf{R}^{3}$$

Cada vez que a fórmula for utilizada, ocorrerá o seguinte:

- a variável R deverá receber um valor **antes** da execução dos cálculos;
- a variável V receberá um valor **após** a execução dos cálculos;
- o valor da variável R será utilizado no cálculo do valor de V;
- a cada nova utilização da fórmula, o valor de R poderá ser alterado e, consequentemente o valor de V também o será.

Segundo (Forsythe,1975), uma variável pode ser vista como uma caixa identificada por um **nome** colocado na tampa desta caixa e por um **valor de uma constante** que poderá estar colocado no interior da caixa. Enquanto o nome sempre permanece o mesmo, o conteúdo da caixa, isto é, a constante nela armazenada pode ser alterada cada vez que a fórmula na qual ela aparece é utilizada.
Embora saibamos que a memória de um computador não é constituída por um conjunto de caixas deste tipo, na prática, ela funciona como se assim fosse.

Segundo (Wirth,1972), uma variável é comparável a um quadro-negro: um valor escrito no quadro-negro pode ser consultado(inspecionado ou lido), tantas vezes quantas se deseja ou pode ser apagado e reescrito. O reescrever, contudo, causa a perda do valor anterior, o qual será substituído pelo novo valor, enquanto as ações de ler ou consultar o valor não provocam a sua perda.

Uma vez conceituados constante, variável e operação pode-se analisar as combinações deles na construção de expressões.
Expressões, no sentido matemático, são representações simbólicas de seqüências de operações a serem feitas sobre determinados operandos, visando a obtenção de um resultado.
As operações representadas nas expressões devem ser válidas para os operandos especificados para que seja possível a obtenção do resultado.
O conjunto de operandos e operações que compõe uma expressão determinam o tipo da expressão e também definem o tipo do resultado que será obtido pela avaliação da expressão. As expressões mais freqüentemente usadas enquadram-se em uma das seguintes categorias:

- Expressões Aritméticas: produzem como resultado um valor numérico;
- Expressões Lógicas : produzem como resultado um valor lógico.

Para cada uma das categorias acima, pode ser definido um conjunto de operações possíveis.
Estas operações podem ser **monádicas** (um operador atua sobre um único operando) ou **diádicas** (um operador atua sobre dois operandos) e, independente da forma de representação, qualquer expressão pode ser decomposta em seqüências encadeadas de operações monádicas e/ou diádicas.

Exemplo1: X - (- Y)

onde o primeiro "-" é uma operação diádica e o segundo "-" é uma operação monádica.

Exemplo2: não(a ou b)

Onde **ou** é uma operação diádica e **não** (que só tem um operando) é uma operação monádica.

As expressões lógicas serão conceituadas e analisadas em outro capítulo.

Uma **Expressão Aritmética** pode ser conceituada como sendo ***" uma constante ou uma variável ou uma combinação de constantes e/ou variáveis por meio de operadores aritméticos "***

Nas expressões aritméticas são usados símbolos para representar:

- operandos : constantes e variáveis;

- operadores : que indicam as operações a serem feitas (+,-,x,/);

- ordem de avaliação: símbolos que indicam alterações na ordem usual de avaliação ((,),] , [, }, }).

Os símbolos também são usados para representar de forma compacta, seqüências longas de operações, tais como:

$$Sn = \sum_{i=1}^{n} \frac{1}{i!}$$ onde $\sum \frac{1}{i!}$ é o somatório dos inversos dos fatoriais de i

Essa representação compacta pode ser substituída por outras, tais como:

$$Sn = \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{n!}$$

É importante notar que a representação compacta não diminui a quantidade de operações a serem feitas, isto é, não minimiza o esforço de cálculo, apenas minimiza o esforço de notação.
A decomposição de uma expressão em passos mais elementares possibilita o emprego de máquinas simples para a solução de problemas de grande complexidade ou grande volume de cálculos repetitivos.
São exemplos de Expressões Aritméticas, de acordo com a definição apresentada acima:

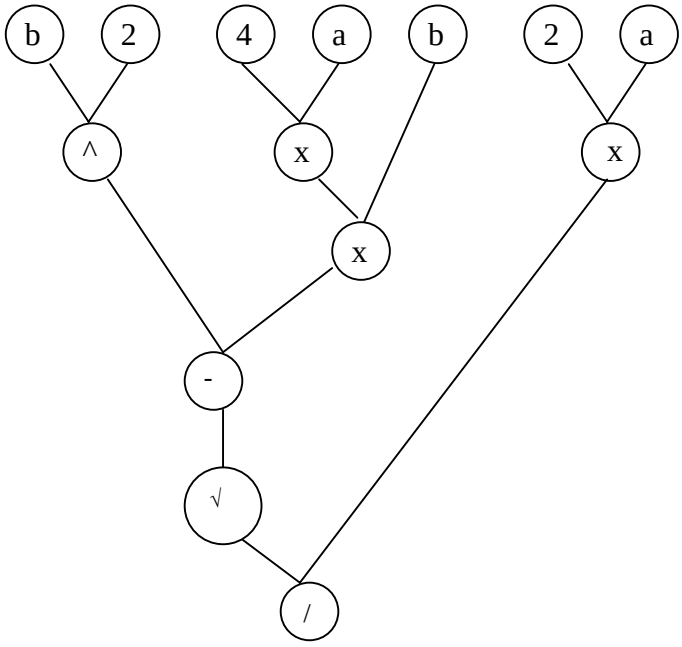
$$50$$
$$a$$
$$b + c$$
$$\frac{b2}{4ac}$$
$$(a + b)^2$$
$$\frac{(-2a + 5b)}{2} + 1$$

Na resolução de uma expressão aritmética deve-se seguir uma ordem pré-definida que depende dos operadores e de símbolos capazes de indicar alteração de ordem de resolução.

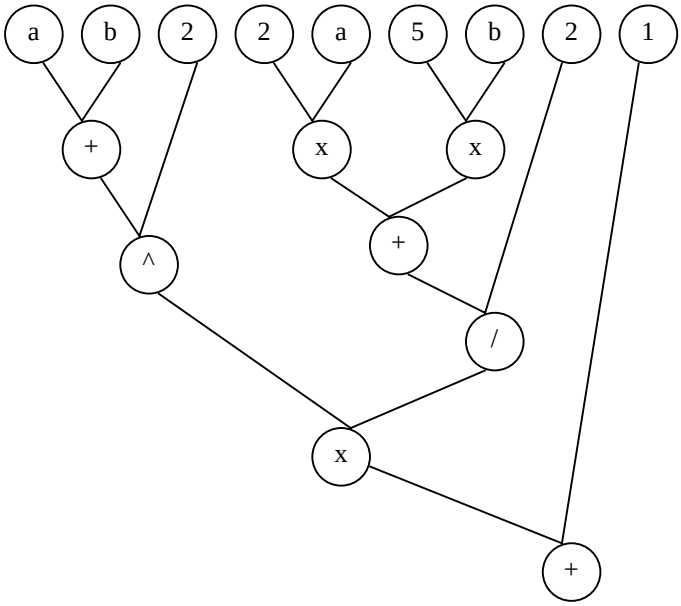
A prioridade natural dos operadores é a seguinte:

1) Potenciações e (-) monádico
2) Multiplicações e divisões
3) Somas e subtrações
4) Parênteses, colchetes ou chaves podem alterar esta ordem
5) Segue-se da esquerda para a direita em caso de indeterminação (mais de uma operação de mesma prioridade).

Exemplo1 : Resolver $\frac{\sqrt{b^2 - 4ac}}{2a}$



Considere-se esta outra expressão: $(a + b)^2 \frac{(2a+5b)}{2} + 1$ que tem a seguinte ordem de operações:



A decomposição de uma expressão deverá atender sempre o seguinte princípio básico: a solução das sub-expressões deve conduzir à solução de toda a expressão.

Tanto o resultado final da expressão quanto os resultados das sub-expressões devem ser armazenados em variáveis.

O ato de armazenar ou escrever um valor em uma variável é chamado de **atribuição de valor**, sendo representado através de símbolos, tais como:

Variável ← valor
Variável := valor
Variável = valor

A atribuição compreende uma ação de substituição do conteúdo de uma variável cujo efeito é: após a atribuição, a variável passa a representar um novo valor, sendo " perdido" o valor anterior.

Assim, uma atribuição tal como:

PI = 3,1416

seguida de outra do tipo:

PI = 3,15

fará com que a variável PI represente o valor 3,15 como resultado atual, perdendo o valor 3,1416 anteriormente nela armazenado.

Considere as atribuições:

A = 5
B = 2
C = 10

Se fizermos em seguida: A = B e B = C, teremos como resultados

A = 2
B = 10
C = 10

Fazendo-se, a seguir, as seguintes atribuições: B = C e A = B, teremos
A = 10
B = 10
C = 10

Outro exemplo nos mostra que se tivermos A = 2 e B = 4 e quisermos intercambiar os valores das variáveis A e B, não poderemos simplesmente escrever:

A = B
B = A

cujo resultado seria

A = 4
B = 4

e não A = 4 e B = 2 como se desejava fazer.

Teremos neste caso que escrever a seguinte seqüência de atribuições:

C = A
A = B
B = C

onde C é uma variável auxiliar que evita perder o conteúdo da variável A que ocorria no primeiro caso. O resultado agora será A = 4, B = 2 e C = 2, sendo que o valor de C não nos interessa mais.

Embora em muitas linguagens algorítmicas (ou de programação) se utilize o símbolo “ = “ como operador de atribuição, deve-se tomar cuidado para não confundi-lo com o operador relacional de igualdade que também usa este símbolo.

Considere a atribuição

N = N + 1

Se o símbolo de “ = “ fosse tomado com o significado de igualdade, a afirmativa seria sempre falsa. Se, entretanto, for tomado como operador de atribuição, e supondo-se que o valor de N seja 5, teremos N = 6, significando “ fazer N ser igual ao valor anterior de N acrescido de uma unidade” ou “ somar uma unidade ao valor atual de N”.

EXERCÍCIOS

1) Conceitue **constante.**
2) Conceitue **variável.**
3) Conceitue **operação.**
4) Qual o **efeito** de uma atribuição de valor a uma variável?
5) Resolva as expressões abaixo para a = 5, b = 1 e c = 2, dando o valor de **x** e a ordem das operações.

$$X = \frac{a^2 - 5b + \frac{c}{2}}{\frac{(b+3}{7}}$$

$$X = \frac{(a+b^2)(a-b) + 4a - 5b + c}{2}$$

6) Conceitue **Expressão Aritmética.**
7) Que são operadores **monádicos e diádicos?**
8) Além das expressões aritméticas que outro tipo de expressões são usadas em algoritmos?
9) Qual a seqüência de operações necessárias para intercambiar os valores de 3 variáveis **a, b, c** de modo que **a** fique com o valor de **b**, **b** fique com o valor de **c** e **c** fique com o valor de **a** ?

10) Se X possui o valor 15 e se forem executadas as seguintes atribuições na ordem em que aparecem:

X = X + 3
X = X – 6
X = X/2
X = 3X Qual será o valor armazenado em X ?

CAPÍTULO II

ALGORITMOS SEQÜENCIAIS

Este capítulo visa conceituar algoritmo, formalizar este conceito, caracterizar algoritmos seqüenciais, bem como, iniciar a apresentação de uma linguagem algorítmica a ser utilizada ao longo do livro.

Nada é mais básico em computação do que o conceito de algoritmo.

" Um conjunto finito de regras, bem definidas, para a solução de um problema em um tempo finito."

Este conceito, aparentemente simples, pode ser formalizado matematicamente com grande precisão, edificando-se sobre a definição toda uma teoria de algoritmos.

Considere o seguinte problema: Dados 3 valores positivos, **a**, **b**, **c**, determinar a sua média aritmética, harmônica, geométrica e ponderada com pesos respectivos de 1, 2 e 3. As tarefas a serem executadas para a solução deste problema, podem ser descritas da seguinte forma:

1) Obter os valores de a, b, c
2) Calcular a média aritmética pela fórmula ma = $\dfrac{a+b+c}{3}$
3) Calcular a média harmônica pela fórmula mh = $\dfrac{3}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}$
4) Calcular a média geométrica pela fórmula mg = $\sqrt[3]{axbxc}$
5) Calcular a média ponderada pela fórmula mp = $\dfrac{1xa+2xb+3xc}{1+2+3}$
6) Comunicar os resultados obtidos: ma,mh,mg,mp
7) Terminar

A seqüência de tarefas acima, especificadas passo a passo, é um algoritmo, pois é formado por um conjunto finito de regras que podem ser consideradas bem definidas e cuja execução produz a solução do problema proposto, após um tempo finito.

A execução da tarefa, neste caso, é corretamente cumprida, se executarmos todos os passos (instruções) na seqüência em que elas aparecem, da primeira até a última, sem omissões e sem repetições.

Um algoritmo com estas características é denominado de **algoritmo seqüencial.**

No passo 1 do algoritmo é indicada a ação de " obter valores ". Para efetuar os cálculos indicados nos passos 2, 3e 4 a pessoa, máquina de calcular ou computador que deverá executar estas tarefas necessita receber os valores de a, b, c. Aos valores recebidos por um algoritmo dá-se o nome de Entradas. Deve-se ter o cuidado de definir claramente o conjunto ou intervalo de conjunto ao qual pertencem as Entradas.

No algoritmo acima, a, b, c podem assumir qualquer valor real, exceto o zero, pois, o inverso de zero não está definido e com isso a média harmônica não poderia ser calculada.

Todo algoritmo possui pelo menos um resultado. O algoritmo do cálculo das médias possui como resultados os valores das médias ma, mh, mg e mp. Os resultados fornecidos por um algoritmo recebem o nome de Saídas.

É importante que num algoritmo cada passo esteja precisamente definido não deixando margem a ambigüidades. Muitos conjuntos de regras encontradas no dia a dia, tais como, receitas, descrições de usos de aparelhos, etc. pecam por definições imprecisas de um ou mais de seus passos.

Assim, uma receita que se refere a uma "pitada de sal" ou "mexa até que fique em ponto de caramelo" não define claramente o quanto isto representa, de forma que cada usuário da receita deve interpretá-lo.

Para precisar o conceito de algoritmo é necessário que ele satisfaça à propriedade da efetividade (não confundir com eficiência), isto é, todas as operações especificadas no algoritmo devem ser suficientemente básicas para que possam ser, pelo menos em princípio, executadas de maneira exata e num tempo finito.

Deve-se observar que na prática não basta que um algoritmo seja de execução finita. Um algoritmo útil precisa terminar não apenas em um tempo finito, mas em um tempo " muito finito ". Por exemplo, dada uma partida de xadrez, existe um algoritmo que determina se as peças brancas podem ou não forçar uma vitória. Eis um algoritmo que pode resolver um problema de grande interesse para um bom número de pessoas. No entanto, não teremos a resposta, pois, tal algoritmo requer um tempo fantasticamente grande, ainda que finito, para chegar a esta conclusão. Não possui a efetividade.

Observe a seguinte lista de instruções, que escreve os números ímpares à partir de 1:

```
Fazer n igual a zero e i igual a 1

Enquanto n for menor do que 5

        Escreve i

        Somar 2 ao i

Terminar
```

A lista acima possui um conjunto finito de passos, claramente definidos, mas sua execução é de tempo infinito, pois, como é fácil ver, o n é feito igual a zero no início e não é alterado por nenhuma instrução. Logo, a instrução " Enquanto " será executada indefinidamente porque n será sempre menor do que 5.

Quando isto ocorre, a lista de instruções é dita " procedimento" e não algoritmo, pois, falta-lhe a condição de efetividade.

Cada passo de um algoritmo pode ser constituído de uma ou mais operações, por exemplo, no procedimento acima a instrução "Enquanto" é constituída de "Escrever i " e " Somar 2 ao i " que são duas operações distintas.

Ao conjunto de duas ou mais operações que devem ser executadas na ordem em que aparecem, denomina-se de **seqüência de operações.**

Na prática não é importante ter-se apenas um algoritmo, mas ter-se um bom algoritmo. O mais importante com respeito a um algoritmo é a sua correção, isto é, se ele resolve realmente o problema proposto e o faz exatamente. Sem esta qualidade, qualquer outra não interessa e o próprio algoritmo não tem sentido de existência.

Outros critérios que devem ser considerados são: o tempo necessário para a sua execução, a adaptabilidade do algoritmo para a solução de problemas similares, sua simplicidade, sua elegância, sua legibilidade, etc. que serão analisados em um capítulo à parte.

Geralmente existe mais de uma seqüência de operações as quais podem ser executadas para se ter a execução de uma determinada tarefa, isto é, existem vários algoritmos para o mesmo problema. Não é, porém, qualquer seqüência de operações que leva à correta execução de uma tarefa. Quanto maior o número de operações para se executar uma tarefa tanto mais algoritmos diferentes existem para executar aquela tarefa.

Considere-se o problema de **encontrar o máximo divisor comum de dois números "m" e "n" inteiros e positivos**:

Solução 1

Seja "r" o resto da divisão de "m" por "n".
($0 \leq r < n$)

Se r = 0 então escrever "n" como resposta e terminar.

Fazer "m" assumir o valor de "n" ,
"n" assumir o valor de "r" e
voltar ao passo inicial.

Solução 2

Se “m” ≥ “n” fazer “m” igual ao resto de “m” dividido por “n” e repetir este mesmo passo.

Trocar os valores de “m” e “n” entre si.

Se “n” = 0 então escrever “m” como resposta e terminar.

Voltar ao passo inicial.

Solução 3

Dividir “m” por “n” e fazer “r” ser o resto.

Se r = 0 escrever “n” como resposta e terminar.

Dividir “n” por “r” e fazer “m” ser o resto.

Dividir “r” por “m” e fazer “n” ser o resto.

Se m = 0 escrever “r” como resposta e terminar.

Se n = 0 escrever “m” como resposta e terminar.

Voltar ao passo inicial.

Solução 4

Se n ≠ 0 então MDC = MDC(n, mod(m,n))
Senão MDC = m.

Onde mod é a função módulo (resto inteiro da divisão de “m” por “n”), Esta solução é a forma recursiva para se calcular o máximo divisor comum entre m e n.

Certamente estas soluções não são as únicas soluções para se resolver este problema.

Quando se dispõe de vários algoritmos para solucionar o mesmo problema, entram em consideração os critérios de escolha do melhor algoritmo. Deve-se, porém, verificar primeiramente se todos os algoritmos são corretos, isto é, resolvem exatamente todo o problema proposto e, somente depois, levam-se em conta os critérios de escolha tais como, tempo de execução, legibilidade, adaptabilidade, elegância, simplicidade, etc.

Para ter-se um algoritmo é necessário:

1) **Que se tenha um número finito de passos.**
2) **Que cada passo esteja precisamente definido, sem possíveis ambigüidades.**
3) **Que existam zero, uma ou mais Entradas tomadas de conjuntos bem definidos.**
4) **Que existam uma ou mais Saídas.**
5) **Que o conjunto de passos nos leve, passo a passo, à execução de uma tarefa útil.**
6) **Que exista uma condição de fim sempre atingida para quaisquer Entradas e num tempo finito.**

Um método para a construção de algoritmos pode, simplificadamente, ser descrito pelos seguintes passos:

1. Ler atentamente o enunciado.
2. Descobrir no enunciado o conjunto de entradas (dados que são fornecidos).
3. Descobrir no enunciado o conjunto de saídas (resultados que devem ser produzidos).
4. Determinar o que deve ser feito para transformar as entradas nas saídas desejadas.
5. Construir o algoritmo.
6. Executar o algoritmo verificando se produz os resultados desejados.

Nos exemplos de algoritmos até agora apresentados foi utilizada uma linguagem usual, sem maiores formalismos. Contudo, para um melhor entendimento dos próprios algoritmos, para poder compará-los com outros algoritmos que foram escritos por colegas e que solucionam o mesmo problema, é, se não necessário, ao menos altamente conveniente formalizar a linguagem algorítmica usada.

Como já visto, um algoritmo geralmente possui uma ou mais entradas. Uma operação de dar entrada de valores será indicada no algoritmo pela seguinte instrução:

Ler (< lista de variáveis >)

Onde **lista de variáveis** é uma variável ou várias variáveis separadas por vírgulas.

Exemplos:
Ler (a)
Ler (a, b, c)
Ler (numero, nome, salário)

Todo algoritmo deve produzir pelo menos um resultado, denominado de saída. Uma operação de saída, no algoritmo, será indicada pela instrução:

Escrever (< lista de expressões >)

Onde lista de expressões é uma constante ou uma variável ou constantes e/ou variáveis e/ou expressões, separadas por vírgulas.

Exemplos:
Escrever (5)
Escrever (x)
Escrever (x, y, z)
Escrever (“ Matrícula = “, m, “ Nota = “, n)
Escrever (“ Faculdade de Informática “)
Escrever ($\frac{a+b}{2}$, 2ab, b^2-4ac)

Num algoritmo existem, geralmente, um ou mais passos indicando cálculos que devem ser realizados. Uma operação de cálculo será indicada pela instrução de atribuição:

Variável = Expressão

Exemplos:
a = 5
x = y
d = $b^2 – 4ac$
nome = “João“

Como, após a execução de um certo número de passos, o algoritmo, deve chegar à condição de tarefa realizada, isto é, termino da execução do algoritmo. Esta operação será indicada pela instrução:

Fim

Como, para se poder construir um algoritmo efetivo, deve-se definir os valores que as variáveis do algoritmo podem assumir, será utilizada a seguinte instrução para indicar os tipos de valores que as variáveis do algoritmo podem assumir:

< Lista de variáveis >: < tipo >

Onde o significado de **lista de variáveis** já foi visto acima e tipo pode ser qualquer um dos seguintes : inteiro, real, caracter, booleano. Por enquanto usaremos somente estes tipos.

Exemplos: n : inteiro
a, b, c : real

Um algoritmo que possui apenas estes tipos de instruções é denominado de **algoritmo seqüencial.**

"Um Algoritmo Seqüencial é aquele cuja execução é efetuada na ordem em que as instruções ocorrem, sem omissão de passos. A passagem de um passo a outro é natural e automática e cada passo é executado uma única vez".

Como exemplo, considere o problema de calcular o fluxo de corrente alternada de um circuito dados: a voltagem, a resistência, a freqüência, a indutância e a capacitância.

As variáveis que representam estas grandezas são:

I = corrente (amperes)
R = resistência (ohms)
C = capacitância (farads)
E = voltagem (volts)
L = indutância (henrys)
F = freqüência (ciclos/seg)

Um algoritmo para resolver este problema é o seguinte:

Algoritmo CalculoCorrente
E, R, F, C, L, I : Real
Início
Ler (E, R, F, L, C)

$$I = \frac{E}{\sqrt{R^2 + [2\pi FL - \frac{1}{2\pi FC}]^2}}$$

Escrever (I)

Fim

O passo 2 do algoritmo acima poderia ser decomposto em diversas instruções de cálculo em lugar de se usar uma só. Independente da decomposição ou não, o algoritmo se resume nas seguintes instruções básicas:

- Ler valores
- Calcular valores
- Escrever resultados
- Parar

Todo algoritmo seqüencial tem exatamente esta estrutura. Portanto, a lógica deste tipo de algoritmo é muito simples e não reside no algoritmo propriamente dito a dificuldade de se escrever algoritmos seqüenciais e sim, no eventual desconhecimento das formas de se calcular os valores desejados.

QUESTIONÁRIO

1) Conceitue algoritmo, enumerando os principais elementos que são significativos neste conceito.
2) Em que consiste a propriedade da **efetividade** de um algoritmo ?
3) Que é um **procedimento** ?
4) Quando é que um algoritmo é dito correto ?
5) Cite 3 outros critérios que podem ajudar na escolha do melhor algoritmo para solucionar um determinado problema.
6) Que é um **algoritmo seqüencial** ?
7) Que tipos de instruções aparecem num algoritmo seqüencial ?
8) Qual a principal dificuldade que pode aparecer em algoritmos seqüenciais ?
9) Por que é interessante formalizar-se a linguagem algorítmica utilizada ?
10) Por que é necessário definir-se os tipos de valores que as variáveis podem assumir ?

EXERCÍCIOS

1) Reescrever o algoritmo que calcula as médias, apresentado no início deste capítulo, utilizando a linguagem apresentada.
2) Escrever um algoritmo que lê o número de um funcionário, seu número de horas trabalhadas, o valor que recebe por hora, o número de filhos com idade menor do que 14 anos e o valor do salário família (pago por filho com menos de 14 anos). Calcular o salário total deste funcionário e escrever o seu número e o seu salário total.
3) Escrever um algoritmo que calcula e escreve o fatorial de 5.
4) Escrever um algoritmo que lê 3 valores reais **a**, **b**, **c** e calcula:
   A área do triângulo que tem **a** por base e **b** por altura;
   A área do círculo de raio **c**;
   A área do trapézio que tem **a** e **b** por bases e **c** por altura;
   A área do quadrado de lado b;
   A área do retângulo de lados **a** e **b**;
   A área da superfície de um cubo que tem **c** por aresta.
5) Escrever um algoritmo que escreve os números ímpares entre 10 e 20.
6) Escrever um algoritmo que lê **p**, **u** e **r** respectivamente o primeiro termo de uma progressão aritmética, o último termo da progressão e a sua razão. Determinar e escrever a soma dos termos desta progressão.
7) Escrever um algoritmo que lê o número de peças do tipo 1, o valor de cada peça do tipo 1, o número de peças do tipo 2, o valor de cada peça do tipo 2 e o percentual do IPI a ser acrescentado. Calcula e escreve o valor total a ser pago por esta compra.
8) Um avião em linha reta, a uma altitude **a** passa sobre um ponto **p** situado no solo, num instante **t** = 0. Se a velocidade é **v**, calcular a distância **d** do avião ao ponto **p** após 30 segundos. Escrever um algoritmo que lê **v** e **a** e calcula e escreve a distância do avião ao ponto **p** após 30 segundos.
9) Uma farmácia paga o seu funcionário a cada sexta-feira e deseja deixar pronto o envelope de pagamento. Escrever um algoritmo que lê o valor do salário do funcionário em cruzeiros e calcula qual o menor número possível de notas de 100, 50, 10, 5 e 1, em que o valor lido pode ser decomposto. Escrever o valor lido e o número de notas de cada tipo que compõe o envelope de pagamento.
10) Escrever um algoritmo que lê o número de um vendedor, o seu salário fixo, o total de vendas por ele efetuadas e o percentual que ganha sobre o total de suas vendas. Calcular o salário total do vendedor e escrever o número e o salário do vendedor.
11) Escrever um algoritmo que lê 3 valores a, b, c que são lados de um triângulo e calcula e escreve a área deste triângulo.

    Área = $\sqrt{s(s-a)(s-b)(s-c)}$

    Onde s = semiperímetro.

12) O custo ao consumidor, de um carro novo, é a soma do custo de fábrica com a percentagem do distribuidor e o percentual dos impostos (aplicados ao custo de fábrica). Escrever um algoritmo que lê o custo de fábrica, o percentual do distribuidor e o percentual dos impostos e calcula e escreve o valor a ser pago pelo consumidor por este carro.
13) Uma revendedora de carros usados paga a seus funcionários vendedores, um salário fixo por mês, mais uma comissão também fixa para cada carro vendido e mais 5% do valor das vendas por ele efetuadas. Escrever um algoritmo que lê o número do vendedor, o número de carros por ele vendidos, o valor de cada carro, o salário

fixo e o valor que recebe por carro vendido e calcula o salário a ser pago a este vendedor, escrevendo o número do vendedor e seu salário total.

14) Considerando que o aumento dos funcionários de uma empresa tenha sido definido da seguinte forma: 80% de um índice chamado INTEMP e mais um percentual de produtividade discutido com a empresa por ocasião do dissídio da categoria. Escrever um algoritmo que lê o número do funcionário, seu salário atual, o valor do INTEMP e o índice de produtividade conquistado e escreve o número do funcionário, seu aumento e seu novo salário.
15) Escrever um algoritmo que lê as coordenadas de dois pontos no plano cartesiano e calcula e escreve a distância entre estes dois pontos, sabendo-se que a fórmula da distância entre dois pontos P1(X1,Y1) e P2(X2,Y2) é

$$\text{Distância} = \sqrt{(X2 - X1)^2 + (Y2 - Y1)^2}$$

16) Escrever um algoritmo que lê 3 valores a, b, c e os escreve. Encontre, a seguir, o maior dos 3 valores e o escreva com a mensagem: " É O MAIOR"

$$\text{Maior} = \frac{a + b + |a - b|}{2}$$

17) Escrever um algoritmo que lê o valor de uma compra em dólares e a taxa do dólar no dia da compra e calcula e escreve o valor a ser pago em reais.
18) Um hotel com 42 apartamentos resolveu fazer promoções para os fins de semana for a da alta temporada, isto é, nos meses de abril, maio, junho, agosto, setembro outubro e novembro. A taxa da promoção é de 22% da diária normal. A expectativa é aumentar a taxa de ocupação de 40 para 70%. Supondo que as expectativas se confirmem, escrever um algoritmo que lê a diária normal e calcula e escreve as seguintes informações:

    a) O valor da diária no período da promoção.
    b) O valor médio arrecadado sem a promoção, durante um mês.
    c) O valor médio arrecadado com a promoção, durante um mês.
    d) O lucro ou prejuízo mensal com a promoção.
19) Escrever um algoritmo que lê a hora de início de um jogo e a hora de fim do jogo, considerando apenas horas inteiras e jogos que começam e terminam no mesmo dia, calcular e escrever o tempo de duração do jogo em horas.
20) O mesmo problema anterior, mas escrevendo o tempo do jogo em minutos.
21) Escrever um algoritmo que lê um número de 3 dígitos e o inverte, escrevendo o número lido e o número invertido.

CAPÍTULO III

ALGORITMOS COM SELEÇÃO

Este capítulo tem por objetivo conceituar **algoritmos com seleção** e **expressões lógicas**.

Os algoritmos seqüenciais podem ser usados na solução de alguns problemas, porém a maioria dos problemas exigem o uso de mais recursos, tais como:

- dados 3 valores a, b, c que representam os coeficientes de uma equação do segundo grau, calcular as raízes reais, somente se o valor do discriminante não for negativo;
- dado um conjunto de valores, determinar quantos deles pertencem ao intervalo [0,50] e quantos pertencem ao intervalo [51,100].

Nos algoritmos, situações como estas são resolvidas através de passos cuja execução está subordinada à veracidade ou não de uma condição.

" Um algoritmo em que se tem a execução de determinados passos subordinada à veracidade de uma condição é denominado de algoritmo com seleção ".

Uma operação de seleção em um algoritmo será representada pelas seguintes instruções:

Se < condição > então < instrução >

Se < condição > então < instrução >
senão < instrução >

Onde **<u>instrução</u>** é qualquer uma das instruções já vistas ou ainda a serem vistas, ou ainda uma seqüência de instruções. No caso de ser uma seqüência, utiliza-se " Início" e " Fim " para agrupar as instruções que formam a seqüência.

Exemplos:

```
Início
   Ler a
   x = a²-1
   Escrever x
Fim
```

```
Início
   x1 = (a+b+c)/3
   x2 = ∛(a.b.c)
Fim
```

<u>Condição</u> é uma expressão lógica.

Para conceituar expressão lógica é conveniente uma explicação sobre o que se entende por álgebra de Boole.

A álgebra de Boole é baseada em proposições. As proposições são assertivas que podem estar corretas (verdadeiras) ou incorretas (falsas). A álgebra booleana é uma álgebra de dois estados: não existem outros estados possíveis além do " verdadeiro" e do " falso ".

Exemplos de proposições:

1. O valor de $\sqrt{2}$ é um número irracional.

2. A equação $Ax^2 + Bx + C = 0$ possui duas raízes reais.

Enquanto a proposição 1. É obviamente verdadeira, a proposição 2. Poderá ser verdadeira ou falsa, dependendo dos valores de A, B, C.

As proposições são combinadas através de inter-relações e é possível demonstrar a veracidade ou a falsidade destas combinações.

As inter-relações entre as proposições são feitas através de operadores lógicos. Um **operador lógico** é um operador cujos operandos são expressões lógicas e cujo resultado é um valor lógico verdadeiro (V) ou falso (F).

Expressões Lógicas

Dados dois valores lógicos quaisquer, T1 e T2, e um operador L, o resultado da expressão lógica **T1 L T2** tem valor verdadeiro (V) ou falso (F) .

Devido ao restrito domínio dos valores lógicos, é possível definir os resultados das operações de forma compacta, através de " tabelas-verdade ". Este tipo de tabela mostra o resultado das operações, para todas as possíveis combinações dos operandos.

A tabela-verdade para os três principais operadores lógicos, a negação, o produto lógico e a soma lógica, pode ser representada como segue:

T1	**T2**	**~T1**	**T1 ^T2**	**T1 ∨T2**
V	V	F	V	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	F

Sendo: T1 e T2 operandos lógicos
~ operador unário de negação (" não ")
^ operador de produto lógico (" e ")
∨ operador de soma lógica (" ou ")

As relações também podem fazer parte das expressões lógicas, como operandos, uma vez que seu resultado é lógico. As relações são definidas entre expressões aritméticas, por meio de operadores relacionais.

Dados dois valores E1 e E2, e um operador relacional R, o resultado da expressão E1 R E2 tem o valor verdadeiro (V) ou falso (F).

Os operadores relacionais são:

- maior (>)
- menor (<)
- igual (=)
- maior ou igual (≥)
- menor ou igual (≤)
- diferente (≠)

Enquanto os operadores lógicos atuam sobre operandos lógicos, os peradores relacionais atuam sobre operandos aritméticos, como pode ser visto na expressão abaixo:

Proposição : X ∈ [1; -1], isto é

" X pertence ao intervalo [1 ; -1] ", cuja expressão lógica correspondente é:

$$\underbrace{X \geq -1}_{R1} \wedge \underbrace{X \leq 1}_{R2}$$

R1 e R2 são relações que fornecem como resultado um valor lógico V ou F. O operador lógico ^ ("e") dará como resultado o valor lógico V somente se ambas as relações R1 e R2 forem verdadeiras.

Os operadores relacionais " ≤ " e " ≥ " atuam sobre os valores X, 1 e –1, que são expressões aritméticas e não lógicas.

Nos algoritmos a correta formulação de condições, isto é, expressões lógicas, é de fundamental importância, visto que as estruturas de **seleção** e também as de **repetição** (que serão vistas adiante), são baseadas nelas.

Portanto, **<u>condição</u>** é

- ou uma **comparação de expressões aritméticas (relações)** por meio de operadores relacionais (=, <, >, ≥, ≤, ≠);
- ou uma **combinação de comparações** por meio de operadores lógicos (~, ^, ∨).

As diversas formulações das condições podem levar a algoritmos distintos. Considere-se o problema:

" Dado um par de valores X, Y , que representam as coordenadas de um ponto no plano, determinar o quadrante ao qual pertence o ponto, ou então, se está sobre algum dos eixos cartesianos ou na origem ".

A solução do problema consiste em determinar todas as combinações de X e Y para as classes de valores positivos, negativos e nulos.

A tabela, a seguir, mostra todas as entradas e saídas possíveis:

Entradas X	Entradas Y	Mensagem de Saída
= 0	= 0	Ponto na origem
= 0	≠ 0	Ponto sobre o eixo Y
≠ 0	= 0	Ponto sobre o eixo X
> 0	> 0	Ponto no Quadrante 1
< 0	> 0	Ponto no Quadrante 2
< 0	< 0	Ponto no Quadrante 3
> 0	< 0	Ponto no Quadrante 4

Os algoritmos podem ser baseados em estruturas **concatenadas** (uma em seqüência a outra) ou em estruturas **aninhadas** (uma dentro da outra), de acordo com a formulação das **condições**.

Algoritmo 3.1: Estruturas concatenadas

```
Algoritmo PosiçãoPonto
x, y : Inteiros
Início
    Ler ( x, y )
    Se x = 0 e y = 0 então escrever ( " Ponto na origem " )
    Se x = 0 e y ≠ 0 então escrever (" Ponto sobre o eixo y " )
    Se x ≠ 0 e y = 0 então escrever (" Ponto sobre o eixo x ")
    Se x > 0 e y > 0 então escrever (" Ponto no quadrante 1 " )
    Se x < 0 e y > 0 então escrever (" Ponto no quadrante 2 " )
    Se x < 0 e y < 0 então escrever (" Ponto no quadrante 3 " )
    Se x > 0 e y < 0 então escrever (" Ponto no quadrante 4 " )
Fim
```

Algoritmo 3.2: Estruturas aninhadas

```
Algoritmo PosiçãoPonto
x, y : Inteiros
Início
    Ler  ( x, y )
    Se x ≠ 0
        então se y = 0
                então escrever  (" Ponto sobre o eixo x " )
                senão se x > 0 então se y > 0 então escrever  ( " Ponto no quadrante 1 " )
                                                  senão escrever  (" Ponto no quadrante 4 ")
                                  senão se y > 0 então escrever  (" Ponto no quadrante 2")
                                                  senão escrever ( " Ponto no quadrante 3 ")
        senão se y = 0 então escrever (" Ponto na origem ")
                          senão escrever (" Ponto sobre o eixo y ")
Fim
```

É importante notar que é possível obter diversos algoritmos para a solução deste problema, através de diferentes combinações de formulações de condições.

As estruturas de seleção aninhadas, geralmente, não são recomendadas por dificultarem o entendimento da solução, como pode ser observado no algoritmo 2 cuja lógica é menos clara do que a do algoritmo 1. Isto não representa problema para o computador, mas para o programador. Quanto mais complexo o algoritmo, mais facilmente ele pode ser escrito de forma incorreta.

Quanto aos tipos de seleção, considere o algoritmo abaixo:

Algoritmo 3.3: Seleção Simples

Algoritmo Medias
i, a, b, c, x : Inteiros
Início
Ler (i, a, b, c)
Se i = 1 então x = $\frac{a+b+c}{3}$
Se i = 2 então x = $\sqrt{a.b.c}$
Se i = 3 então $x = \frac{3}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}$
Se i > 3 ou i < 1 então x = 0
Escrever (x)
Fim

O tipo de seleção mostrado no exemplo 3 é denominado de **seleção simples,** pois, decide entre a execução ou não de uma ou mais instruções conforme a condição for verdadeira ou falsa.

Algoritmo 3.4: Seleção Dupla

Algoritmo Raizes
a, b, c, x, x1, x2, d : Reais
Início
Ler (a, b, c)
Se a = 0 então se b ≠ 0 então Início
x = -c/b
Escrever (x)
Fim
senão Escrever (c)
senão Início
d = b^2 - 4ac
Se d < 0 então Escrever (“ Equação sem raizes reais “)
senão Início
x1 = $\frac{-b+\sqrt{d}}{2a}$
x2 = $\frac{-b-\sqrt{d}}{2a}$
Escrever (x1, x2)
Fim
Fim
Fim

Neste caso o algoritmo sempre seleciona entre duas ações ou conjuntos de ações, executando uma delas e desprezando a outra, conforme a condição for verdadeira ou falsa.

Assim, se **a = 0 e b ≠ 0** calcula e escreve a raiz da equação do primeiro grau.
Se **a = 0 e b = 0** escreve somente o valor de c
Se **a ≠ 0 e b^2- 4ac < 0** escreve a mensagem " Equação sem raízes reais "
Se **a ≠ 0** e **b^2- 4ac > 0** calcula e escreve as raízes da equação do segundo grau que tem **a, b, c,** por coeficientes.
Este tipo de seleção é denominado de **seleção dupla** .

Seleção Múltipla

Embora os algoritmos com seleção múltipla possam ser resolvidos com o uso de instruções de seleções simples, uma forma mais elegante e que tem correspondência nas linguagens de programação é o uso da instrução **Case**, cuja forma é a seguinte:

```
Case < expressão aritmética > com
    < val1 > : instrução 1
    < val2 > : instrução 2
    < val3 > : instrução 3
        .          .
        .          .
        .          .
    < valn > : instrução n
    Senão  instrução
Fim_Case
```

Onde < val1 >, < val2 >, < val3 >, ... , < valn > são valores assumidos pela expressão aritmética. Para cada valor pode-se especificar uma ação ou um conjunto de ações a serem executadas.

Algoritmo 3.5: Seleção Múltipla

Algoritmo Médias2
i : inteiro
a, b, c, x : real
Início
Ler (i, a, b, c)
Case i com
1: x = $\frac{a+b+c}{3}$
2: x = $\sqrt[3]{axbxc}$
3: x = $\frac{3}{1/a + 1/b + 1/c}$
Senão x = 0
Fim_Case
Escrever (x)
Fim

ou

Algoritmo Médias 3
i : inteiro
a, b, c, ma, mg, mh : real
Início
Ler (i, a, b, c)
Case i com
1: Início
ma = $\frac{a+b+c}{3}$
Escrever ma
Fim
2: Início
mg = $\sqrt[3]{axbxc}$
Escrever (mg)
Fim
3 : Início
mh = $\frac{3}{1/a + 1/b + 1/c}$
Escrever (mh)
Fim
Senão escrever ("Código Inválido")
Fim_Case
Fim

Com a introdução das instruções de seleção, o número de algoritmos que podem ser construídos como solução para um problema já aumenta consideravelmente.

Considere-se o problema de " ler 3 valores a, b, c, quaisquer e encontrar o maior dos 3 escrevendo-o com a mensagem: È O MAIOR."

Algoritmo 3.6 :Versão 1

Algoritmo Maior
a, b, c, mab, mabc : Reais
Início
Ler (a, b, c)
mab = $\dfrac{a+b+|a-b|}{2}$
mabc = $\dfrac{mab+c+|mab-c}{2}$
Escrever (mabc, " E O MAIOR ")
Fim

Algoritmo 3.7 :Versão 2

Algoritmo Maior
a, b, c : Reais
Início
Ler (a, b, c)
Se a ≥ b e a ≥ c então escrever (a, " É O MAIOR ")
Se b > a e b ≥ c então escrever (b, " É O MAIOR ")
Se c > a e c > b então escrever (c, " É O MAIOR ")
Fim

Algoritmo 3.8 :Versão 3

Algoritmo Maior
a, b, c : Reais
Início
Ler (a, b, c)
Se a ≥ b e a ≥ c então escrever (a, " É O MAIOR ")
senão Se b > a e b ≥ c então escrever (b, " É O MAIOR ")
senão escrever (c, " É O MAIOR ")
Fim

Algoritmo 3.9 :Versão 4

Algoritmo Maior
a, b, c : Reais
Início
Ler (a, b ,c)
Se a > b então Se a > c então escrever (a, " É O MAIOR ")
senão escrever (c, " É O MAIOR ")
senão Se b > c então escrever (b, " É O MAIOR ")
senão escrever (c, " É O MAIOR ")
Fim

Algoritmo 3.10 :Versão 5

```
Algoritmo Maior
a, b, c, maior : Reais
Início
    Ler ( a, b, c )
    maior = a
    Se maior < b então maior = b
    Se maior < c então maior = c
    Escrever ( maior, “ É O MAIOR ” )
Fim
```

As versões 2, 3 e 4, se aumentarmos o número de variáveis para 4 ou 5 já complicariam bastante. Sugere-se reescrever alguma destas versões para 4 valores.

A versão 5 já não se torna complexa para até 10 valores. Contudo, se o número de valores crescer muito outras soluções devem ser construídas, como se verá mais adiante.

As diferentes versões anteriormente apresentadas (5 da muitas possíveis), mostram que um problema admite geralmente uma quantidade de algoritmos que são caminhos para se chegar à solução do problema. Isto dá margem ao uso de criatividade e enseja análises sobre qual dos algoritmos é uma solução melhor. Isto será discutido mais adiante.

QUESTIONÁRIO

1. Conceitue **algoritmo com seleção**.
2. Apresente as formas de representação de uma instrução de seleção em um algoritmo.
3. Conceitue **expressão lógica ou condição**.
4. Os operadores relacionais atuam sobre que tipo de operandos ?
5. Os operadores lógicos atuam sobre que tipo de operandos ?
6. Que se entende por seleção simples ?
7. Que significa seleção dupla ?

EXERCÍCIOS

1) Escrever um algoritmo que lê 3 valores **a**, **b**, **c** e calcula e escreve a média ponderada com peso 5 para o maior dos 3 valores e peso 2.5 para os outros dois.
2) Escrever um algoritmo que lê 3 valores **a**, **b**, **c** e verifica se eles formam ou não um triângulo. Supor que os valores lidos são inteiros e positivos. Caso os valores formem triângulo calcular e escrever a área deste triângulo. Se não formarem triângulo, escrever os valores lidos com a mensagem: " Não formam triângulo ".
3) Escrever um algoritmo que lê dois valores **a**, **b** e os escreve com a mensagem: " São Múltiplos" ou " Não são Múltiplos ".
4) Escrever um algoritmo que lê um conjunto de 4 valores **i**, **a**, **b**, **c,** onde i é um valor inteiro e positivo e **a, b, c** são quaisquer valores reais. Escreva os valores lidos e, a seguir :
   Se i = 1 escrever os três valores a, b, c em ordem crescente.
   Se i = 2 escrever os três valores a, b, c em ordem decrescente.
   Se i = 3 escrever os três valores a, b, c de forma que o maior fique entre os outros dois.
5) Escrever um algoritmo que lê um conjunto de 6 valores X1, X2, Y1, Y2, Z1, Z2, que representam as coordenadas cartesianas de 3 pontos P1(X1,X2), P2(Y1,Y2) e P3(Z1,Z2). Calcule as distâncias entre P1 e P2, entre P2 e P3 e entre P1 e P3. Se os segmentos de retas calculados formam um triângulo, calcular e escrever a área deste triângulo, caso contrário, escrever as distâncias calculadas.
6) Escrever um algoritmo que lê o número de um vendedor de uma empresa, seu salário fixo e o total das vendas por ele efetuadas. Sabe-se que cada vendedor recebe um salário fixo, mais uma comissão proporcional às vendas por ele efetuadas. A comissão é de 3% sobre o total das vendas até 10.000,00 e 5% sobre o que ultrapassa este valor. Escrever o número do vendedor, o total de suas vendas, seu salário fixo e seu salário total.
7) Escrever um algoritmo que lê 3 comprimentos de lados a, b, c e os coloca em ordem decrescente, de forma que o a represente o maior dos 3 valores. Determine, a seguir, o tipo de triângulo que estes 3 lados formam com base nas seguintes relações, escrevendo sempre os valores lidos e a mensagem adequada.
   Se $a > b + c$ então não formam triângulo algum;
   Se $a^2 = b^2 + c^2$ então formam um triângulo retângulo;
   Se $a^2 > b^2 + c^2$ então formam um triângulo obtusângulo;
   Se $a^2 < b^2 + c^2$ então formam um triângulo acutângulo;
   Se $a = b$ e $b = c$ então formam um triângulo equilátero;
   Se $a = b$ ou $b = c$ ou $a = c$ e $a \neq b$ ou $a \neq c$ então formam triângulo isósceles.
8) O departamento que controla o índice de poluição do meio ambiente mantém 3 grupos de indústrias que são altamente poluidoras do meio ambiente. O índice de poluição aceitável varia de 0.05 até 0.25. Se o índice sobe para 0.3 as indústrias do grupo 1 são intimadas a suspenderem as suas atividades. Se o índice cresce para 0.4 as indústrias dos grupos 1 e 2 recebem a intimação de suspensão das atividades. Se o índice atingir 0.5 então todos os 3 grupos são notificados.
   Escrever um algoritmo que lê o índice de poluição medido e emite as notificações adequadas aos diferentes grupos de indústrias.
9) Escrever um algoritmo que lê a hora de início e a hora de fim do jogo (considerando apenas horas inteiras) e calcula e escreve a duração do jogo em horas, sabendo-se que a duração máxima para o jogo é de 24 horas e que o jogo pode iniciar em um dia e terminar no dia seguinte.
10) Escrever um algoritmo que lê o número de um funcionário, o número de horas por ele trabalhadas, o valor que recebe por hora, o número de filhos com menos de 14 anos, a idade do funcionário, o tempo de serviço do funcionário e o valor do salário família por filho.
    Calcular o salário bruto, o desconto do INSS (8.5% do salário bruto), e o salário família.
    Calcular o desconto do Imposto de Renda na fonte como segue:
    Se Salário Bruto > 1500,00 então Imposto de Renda = 15% do Salário Bruto;
    Se Salário Bruto > 500,00 e Salário Bruto ≤ 1500,00 então Imposto de Renda = 10% do Salário Bruto
    Se Salário Bruto < 500,00 então Imposto de Renda = 0.
    Calcular o adicional conforme especificado:
    Se idade > 40 anos então adicional = 2% do Salário Bruto;
    Se tempo de serviço > 15 anos então adicional = 3.5% do Salário Bruto;

Se tempo de serviço < 15 anos mas superior a 5 anos e idade > 30 anos então adicional = 1.5% do Salário Bruto.

Calcular o Salário Líquido

Escrever o número do funcionário, seu salário bruto, o total de seus descontos, o adicional e o salário líquido.

11) Escrever um algoritmo que lê o número de identificação de um aluno e as 3 notas obtidas por este aluno nas 3 verificações ao longo do semestre, bem como, a média dos exercícios que fazem parte da avaliação. Para cada aluno, calcular a média de aproveitamento, usando a fórmula:

$$MA = \frac{N1 + N2x2 + N3x3 + ME}{7}$$

A atribuição de conceitos obedece à tabela abaixo:

Média de Aproveitamento	Conceito
≥ 9.0	A
≥ 7.5 e < 9.0	B
≥ 6.0 e < 7.5	C
≥ 4.0 e < 6.0	D
< 4.0	E

O algoritmo deve escrever o número do aluno, suas notas, a média dos exercícios, a média de aproveitamento, o conceito correspondente e a mensagem " APROVADO" se o conceito for A, B ou C e " REPROVADO" se o conceito for D ou E.

12) A empresa XYZ decidiu conceder um aumento de salários a seus funcionários de acordo com a tabela abaixo:

SALÁRIO ATUAL	ÍNDICE DE AUMENTO
0 - 400	15%
401 – 700	12%
701 – 1000	10%
1001 – 1800	7%
1801 – 2500	4%
Acima de 2500	Sem Aumento

Escrever um algoritmo que lê, o número de um funcionário e o seu salário atual e escreve o número do funcionário, seu salário atual, o percentual de seu aumento e o valor do salário corrigido.

13) Escrever um algoritmo que lê a hora de início de um jogo e a hora de término do jogo, ambas subdivididas em 2 valores distintos, a saber: horas e minutos. Calcular e escrever a duração do jogo, também em horas e minutos, considerando que o tempo máximo do jogo é de 24 horas, que sua duração não pode ser nula e que o jogo pode começar em um dia e terminar no dia seguinte.

14) Escrever um algoritmo que lê um código **i** e 3 valores **a**, **b**, **c**. Tanto o código **i**, quanto os valores **a**, **b**, **c**, lidos são inteiros e positivos.

Se código = 1 então calcular a área do trapézio de bases **a** e **b** e altura **c** e escrevê-la juntamente com os valores lidos.

Se código = 2 então se **a**, **b**, **c**, formam equação de segundo grau com raízes reais então calcular e escrever estas raízes.

Se código = 3 então calcular a média geométrica dos 3 valores e escrevê-la juntamente com os valores lidos.

Se código = 4 então se **a**, **b**, **c**, formam triângulo, calcular e escrever a área deste triângulo.

Se código > 4 então escrever " código inválido".

15) Escrever um algoritmo que lê um valor m.

Se m não é inteiro escrever a mensagem " O VALOR DE M NÃO É INTEIRO ".

Se m é inteiro e negativo escrevê-lo com a mensagem " É NEGATIVO ".

Caso contrário verificar se m é par ou ímpar e escrevê-lo com a mensagem correspondente.

16) Um supermercado vende cerveja de três tipos (c1, c2 e c3). Como o supermercado deseja fazer uma promoção destas 3 marcas de cerveja e atender, ao mesmo tempo, um bom número de clientes, o número de cervejas de cada tipo foi limitado por cliente em 18 do tipo c1, 12 do tipo c2 e 6 do tipo c3.

Escrever um algoritmo que lê a quantidade de cervejas do tipo c1, o valor unitário da cerveja c1, a quantidade de cervejas do tipo c2, o valor unitário da cerveja do tipo c2, a quantidade de cervejas do tipo c3 e o valor unitário de cada cerveja do tipo c3. O algoritmo deve considerar as limitações impostas pelo supermercado,

isto é, se o cliente pediu mais unidades do que o permitido deve-lhe ser debitado apenas o limite permitido. Deverá escrever o número de itens de cada tipo efetivamente fornecido e o valor total a ser pago.

17) Escrever um algoritmo que lê um valor inteiro e positivo m , de 3 dígitos e escreve o valor lido e o seu equivalente em algarismos romanos (I , V, X, L, C, D, M).

18) Escrever um algoritmo que lê os nomes de 3 produtos, o preço unitário de cada um e a quantidade de cada produto que foi solicitada.
O algoritmo deverá escrever:
   a) O nome dos produtos cujo preço é superior a 50 reais;
   b) O nome e o preço dos produtos que custam menos do que 30 reais;
   c) O preço médio dos 3 produtos;
   d) O nome e o preço dos produtos cujo valor é superior ao preço médio;
   e) O valor total a ser pago pelo pedido.

CAPÍTULO IV

ALGORITMOS COM REPETIÇÃO

Este capítulo tem por objetivo conceituar algoritmo com repetição, introduzir as noções de variável contador, variável acumulador e apresentar os tipos de instrução de repetição geralmente utilizados em algoritmos que repetem uma ou mais instruções.

Na prática, um algoritmo não é executado para um único conjunto de valores conforme visto em algoritmos seqüenciais e em algoritmos com seleção.

Em algoritmos deste tipo, aparecem com freqüência, conceitos tais como: variável contador e variável acumulador.

Uma **variável contador** é uma variável qualquer que recebe um valor inicial (geralmente 0) e é incrementada em algum outro ponto do algoritmo, de um valor constante (geralmente 1).

Exemplo:

.
.
.
Cont = 0
.
.
.
Cont = Cont + 1

O significado de **Cont = 0** é armazenar, na posição de memória de nome Cont o valor 0. Enquanto o significado de **Cont = Cont + 1** é armazenar na posição de memória Cont, o valor que já estava nesta posição acrescido de uma unidade.

Suponha que se queira calcular e escrever a média aritmética das 3 notas n1, n2 e n3, obtidas por cada um dos 30 alunos de uma turma. Para cada aluno será lido o seu número e suas 3 notas.

Para este tipo de algoritmo necessita-se de uma nova instrução, a saber:

```
Enquanto < condição >
    Início
            < seqüência de instruções >
    Fim_Enquanto
```

Onde **seqüência de instruções** são pelo menos duas outras instruções quaisquer.

Algoritmo 4.1:

```
Algoritmo Média
Num, Cont : Inteiros
N1, n2, n3, Media : Reais
Início
    Cont = 0
    Enquanto Cont < 30
        Início
            Ler ( num, n1, n2, n3 )
            Media = (n1 + n2 + n3) / 3
            Escrever ( num, Media )
            Cont = Cont + 1
        Fim_enquanto
Fim
```

Suponha agora que se queira, no final saber a média geral desta turma. Aí necessitamos de outro conceito que é o de variável acumulador.

Uma **variável acumulador** é qualquer variável que recebe um valor inicial constante (geralmente 0) e é incrementada em algum outro ponto do algoritmo de um valor variável.

Exemplo:

.
.
Soma = 0
.
.
.
Soma = Soma + Valor

O significado de **Soma = 0** é armazenar no endereço de memória cujo nome é Soma o valor 0. Enquanto o significado de **Soma = Soma + Valor** é armazenar na posição de memória cujo nome é Soma, o valor que já está nesta posição acrescido do valor da variável Valor.

Algoritmo 4.2:

```
Algoritmo MediaGeral
Num, Cont : Inteiros
Soma, Media, n1, n2, n3, Medgeral: Reais
Início
    Soma = Cont  = 0
   Enquanto Cont < 30
       Início
           Ler ( num, n1, n2, n3 )
           Media = (n1 + n2 + n3) / 3
           Escrever ( num, Media )
           Soma = Soma + Media
           Cont = Cont + 1
       Fim_Enquanto
   Medgeral = Soma/30
   Escrever ( Medgeral )
Fim
```

Em ambos os algoritmos têm-se um conjunto de instruções que é repetido um certo número de vezes, isto é o que caracteriza um algoritmo com repetição.

"Todo algoritmo que tem a execução de um ou mais de seus passos repetido duas ou mais vezes é denominado de algoritmo com repetição".

O número de vezes que um certo conjunto de instruções é repetido pode ser previamente conhecido, como no caso dos exemplos anteriores, ou pode não ser conhecido com antecedência.

Suponha o mesmo problema do algoritmo 4.2, apenas não se conhecendo o número de alunos da turma ou querendo se escrever uma solução que sirva para uma turma com qualquer número de alunos.

Neste caso deve-se encontrar uma condição de fim para o algoritmo, isto é, quando deve parar a repetição.

No caso deste problema não é difícil achar-se esta condição de fim. Como sabemos que o número do aluno certamente não será um valor negativo, podemos repetir a leitura e as demais instruções até que seja lido um número de aluno negativo, por exemplo.

Algoritmo 4.3:

```
Algoritmo MediaGeral
Cont, Num :Inteiros
Soma, n1, n2, n3, Media, Medgeral : Reais
Início
    Soma = Cont = 0
    Ler ( Num, n1, n2, n3 )
    Enquanto  Num > 0
        Início
            Media = (n1 + n2 + n3) / 3
            Soma = Soma + Media
            Cont = Cont + 1
            Ler ( Num, n1, n2, n3 )
        Fim_enquanto
    Medgeral = Soma/Cont
    Escrever ( Medgeral )
Fim
```

Como no algoritmo 4.3, na maioria dos casos interessam soluções genéricas para os algoritmos.

O teste de valor final pode apresentar-se sob duas formas:

a) Se os valores lidos pertencem a um intervalo bem conhecido ou a algum conjunto bem restrito, adiciona-se, no final do conjunto de valores válidos, um valor que está for a do intervalo ou do conjunto restrito utilizado no algoritmo.

   Exemplo1: Ler um conjunto não determinado de números de alunos.

   Ler Num
   Se Num < 0 então escrever " Fim "

   Como os números de alunos costumam ser inteiros e positivos, coloca-se no final, para terminar, um número negativo.

   Exemplo 2: Ler um conjunto não conhecido de valores inteiros para m, um de cada vez.

   Ler m
   Se m ≠ int(m) então escrever " Terminou a leitura do conjunto "

b) Se os valores lidos podem assumir qualquer valor (inteiro, real, positivo, negativo) então não há como fornecer um valor diferente. Lembre-se de que as variáveis geralmente não misturam caracteres com números (ou é numérica e representa um número, ou é alfanumérica e representa caracteres).
   Neste caso, em vez de ler x (que pode ser qualquer valor numérico), pode-se ler x e y onde para y será fornecido 0 quando o valor de x interessa e 1 para terminar.

   Exemplo 1: Ler um número qualquer de ternos de valores a, b, c todos reais.

   Ler a, b, c, i
   Se i = 1 então escrever " Terminar "

   Como a, b, c podem ser quaisquer valores numéricos, não há como se basear neles para terminar a leitura. Por isso, foi adicionada uma variável nova I, para a qual se fornecerá 0 enquanto a leitura deve continuar e 1 quando se deseja terminar.

Isto está ilustrado abaixo :

a	b	c	i
5	-1	5.23	0
-6.12	4	9.8	0
44	-211	1	0
0	1	-1	0
1000	0	-8	0
0	0	0	1

Exemplo 2: Ler um conjunto não conhecido de valores para x, $x \in R$.

Ler x, L
Se L então escrever " Acabaram os valores de x "

Supõe que a variável L (variável lógica: verdadeiro ou falso) recebe o valor falso enquanto nem todos os valores de x tiverem sido lidos e recebe o valor verdadeiro quando acabaram os valores de x.

X	L
0.2345	falso
-1	falso
-45	falso
1433	falso
0.2346	falso
0	verdadeiro

Como na maioria das linguagens de programação existem várias instruções diferentes de repetição, será apresentada outra forma para uma instrução de repetição.

```
Para variável = Exp1 até  Exp2  [ inc  Exp3 ]
    Início
        < Instruções >
    Fim_Para variável
```

Onde variável é qualquer variável, neste caso, denominada de **variável de controle da repetição**.
Exp1, Exp2 e Exp3 são expressões aritméticas.
Exp1 representa o valor inicial da variável de controle.
Exp2 representa o valor final da variável de controle.
Exp3 representa o incremento da variável de controle.

Se o valor da variável de controle for 1 não será necessário especificá-lo. Por isso [inc Exp3] aparece entre colchetes, indicando que a sua especificação é opcional 9 só quando necessário).

Numa instrução deste tipo estão implícitas 3 operações, a saber :

1. Uma inicialização da variável de controle com um valor inicial igual a Exp1 .
2. Um teste que verifica se o valor da variável de controle já ultrapassou o seu valor final indicado por Exp2
3. A cada nova repetição ocorrerá um incremento da variável de controle de um valor igual a 1 ou de um valor igual a Exp3.

Se ao fazer o teste é verificado que o valor da variável de controle já ultrapassou o valor final permitido, a repetição é encerrada e o controle é automaticamente transferido para a primeira instrução depois de " Fim_Para ".

Exemplos de instruções de repetição deste tipo:

a) Para i = 1 até 5
    Início
        Instruções
    Fim_Para i

b) Para x = a até b inc 2
    Início
        Instruções
    Fim_Para x

c) Para i = n até 1 inc –1
    Início
        Instruções
    Fim_Para i

d) Para a = $\frac{x+j}{2}$ até $b^2 - 4 + cd$ inc c/2
    Início
        Instruções
    Fim_Para a

e) Para x = 0.5 até 17.3 inc 1.3
    Início
        Instruções
    Fim_Para x

No exemplo c) quando o incremento é negativo, o teste é sempre se o valor da variável de controle já é **menor do que** o valor final especificado. Nos demais casos o teste é sempre **maior do que.**

Algoritmo 4.4 : Verificar se um número m lido é primo.

```
Algoritmo Divisores
c,m,i : Inteiros
Início
   Ler ( m )
   Enquanto m >0
       Início
         c = 2
         Para i = 2 até m/2
             Se m/i = int ( m/i ) então c = c + 1
         Fim_Para i
         Se c > 2 então escrever ( m, "possui" c, "divisores")
                  senão escrever ( m, " É PRIMO ")
         Ler ( m )
       Fim_enquanto
Fim
```

Obs.: Neste algoritmo foram usadas as duas instruções de repetição apresentadas. Quando não é conhecido o número de valores ou conjuntos de valores que devem ser lidos não é possível utilizar a repetição do tipo "Para". A repetição do tipo " Enquanto" é mais geral e sempre pode ser usada.

Se tivermos dentro de uma repetição, seja de qual tipo for, uma outra repetição, esta última deverá estar inteiramente contida na primeira.

Considere o seguinte problema: ***Escrever um algoritmo que lê 10 valores para m, todos inteiros e positivos, e, para cada m lido, calcula o fatorial de m, escrevendo-o juntamente com o valor lido.***

Algoritmo 4.5 Cálculo do Fatorial

```
Algoritmo Fatorial
c,m, i, fat : Inteiros
Início
    Para c = 1 até 10
        Início
            Ler ( m )
            fat = 1
            Para i = 1 até m
                fat = fat x i
            Fim_Para i
            Escrever ( m, fat )
        Fim_Para c
Fim
```

Algoritmos com repetições dentro de outras repetições são denominados de algoritmos com repetições aninhadas.

QUESTIONÁRIO

1. Que é uma **variável Contador** ?
2. Que é uma **variável Acumulador** ?
3. Qual a diferença entre uma variável Contador e uma variável Acumulador ?
4. Que se entende por **teste do valor final** e quando é usado ?
5. Conceitue **algoritmo com repetição**.
6. Que são repetições aninhadas ?

EXERCÍCIOS

1. Escrever um algoritmo que lê 5 valores para a, um de cada vez, e conta quantos destes valores são negativos, escrevendo esta informação.
2. Escrever um algoritmo que gera e escreve os números ímpares entre 100 e 200.
3. Escrever um algoritmo que lê 10 valores, um de cada vez, e conta quantos deles estão no intervalo [10, 20] e quantos deles estão fora deste intervalo, escrevendo estas informações.

4. Escrever um algoritmo que lê um número não conhecido de valores, um de cada vez, e conta quantos deles estão em cada um dos intervalos [0, 25), [25, 50), [50, 75) e [75, 100], escrevendo estas informações.

5. Escrever um algoritmo semelhante ao anterior, que calcula as médias aritméticas de cada intervalo e as escreve, juntamente com o número de valores encontrados em cada intervalo.

6. A série de Fibonacci tem como dados os 2 primeiros termos da série que são respectivamente 0 e 1. À partir deles, os demais termos são construídos pela seguinte regra:

$$t_n = t_{n-1} + t_{n-2}$$

Escrever um algoritmo que gera os 10 primeiros termos desta série e calcula e escreve a sua soma.

7. Escrever um algoritmo que gera os 10 primeiros termos da série de Fibonacci e os escreve, juntamente com o seu número de ordem.

8. Escrever um algoritmo que gera os 30 primeiros termos da série de Fibonacci e escreve os termos gerados com a mensagem: “ É PRIMO” ou “ NÃO É PRIMO “, conforme o caso.

9. Escrever um algoritmo que lê um conjunto não determinado de valores, um de cada vez, e escreve uma tabela, contendo 20 linhas em cada página. A tabela deve conter o valor lido, seu quadrado, seu cubo e sua raiz quadrada.

10. Escrever um algoritmo que lê um número não determinado de valores para m, todos inteiros e positivos, um de cada vez. Se m for par, verificar quantos divisores possui e escrever esta informação. Se m for ímpar e menor do que 12 calcular e escrever o fatorial de m. Se m for ímpar e maior do que 12, calcular e escrever a soma dos inteiros de 1 até m.

11. Escrever um algoritmo que lê um número não determinado de valores a, todos inteiros e positivos, um de cada vez, e calcule e escreva a média aritmética dos valores lidos, bem como, a quantidade de valores pares, a quantidade de valores ímpares, a percentagem dos valores pares e a percentagem dos valores ímpares.

12. Escrever um algoritmo que escreve os números primos entre 100 e 200, bem como, a soma destes números primos.

13. Escrever um algoritmo que lê 10 valores para **n**, um de cada vez, todos inteiros e positivos, e para cada **n** lido, escreva a tabuada de **1** até **n** de **n**.

1 x n = n 2 x n = 2n . . . . n x n = n^2

14. Escrever um algoritmo que lê 5 pares de valores **a, b** , todos inteiros e positivos, um par de dada vez, com **a** < **b** e escreve os inteiros pares de **a** até **b**, incluindo o **a** e **b** se forem pares.

15. Escrever um algoritmo que lê um número não determinado de pares de valores **m, n** , todos inteiros e positivos, um par de cada vez, e calcula e escreve a soma dos **n** inteiros consecutivos à partir de **m** inclusive.

16. Escrever um algoritmo que lê um número não determinado de conjuntos de valores, cada um formado pelo número do aluno e suas 3 notas. Calcular, para cada aluno, a média ponderada com pesos respectivos de 4.0 para a maior nota e peso 3 para as outras duas. Escrever o número do aluno, suas 3 notas, a média calculada e uma mensagem " APROVADO " se a nota for ≥ 5 ou " REPROVADO" se nota < 5.

17. Escrever um algoritmo que lê um valor **n** inteiro e positivo e que calcula e escreve o valor de E, onde :

$$E = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{n!}$$

18. Um Clube de Futebol de uma cidade fez uma pesquisa entre seus sócios, coletando dados sobre o salário e o número de filhos dos sócios. O Clube deseja saber:

    a) A média do salário dos sócios

    b) A média do número de filhos

    c) O maior salário

    d) O percentual de pessoas com salário até R$ 400,00

    O final da leitura de dados se dará quando da leitura de um salário negativo.

19. Gustavo tem 1,40 metros e cresce 15 centímetros por ano, enquanto Juliano tem 1,10 metros e cresce 3 centímetros por ano. Construa um algoritmo que calcula e escreve, quantos anos serão necessários para que Juliano ultrapasse Gustavo.

20. Em uma eleição presidencial existem 4 candidatos. Os votos são informados por meio de códigos. Os dados utilizados para a contagem dos votos obedecem à seguinte codificação:

    Os códigos 1, 2, 3 e 4 representam os votos para os respectivos candidatos.

    O código 5 representa o voto nulo.

    O código 6 representa o voto em branco.

    Elabore um algoritmo que lê os votos, um de cada vez, e calcula e escreve:

    - o total de votos para cada candidato

- o total de votos nulos

- o percentual de votos em branco

21. Escrever um algoritmo que lê um conjunto de 15 valores, um de cada vez, acompanhados de um código 1 ou 2. O valor representa o número de cobaias utilizadas em cada uma das 15 experiências feitas, e os códigos 1 ou 2 representam respectivamente coelhos e ratos. Quer-se saber o número total de cobaias utilizadas, o total de coelhos, o total de ratos, a percentagem de coelhos e a percentagem de ratos. Escrever estes valores.

22. Escrever um algoritmo que lê, para cada um dos vendedores de uma empresa, o seu número de identificação, seu salário fixo e o total das vendas por ele efetuadas em reais. Cada vendedor recebe um salário fixo e uma comissão proporcional às vendas por ele efetuadas. A comissão é de 3% sobre o total de vendas até 10000,00 e 5% sobre o que ultrapassa este valor. Escrever, para cada vendedor, o seu número de identificação, o seu salário fixo e o seu salário total.

23. Escrever um algoritmo que lê um número **n** que representa o número de termos de uma Progressão Aritmética, $\mathbf{a_1}$, o primeiro termo desta progressão e **r** , razão desta progressão. Escrever os **n** termos desta progressão, bem como, a sua soma.

24. Escrever um algoritmo que lê 5 conjuntos de 2 valores, o primeiro representando o número de um aluno, e o segundo representando a sua altura em centímetros. Encontrar o aluno mais alto e o mais baixo e escrever seus números, suas alturas e uma mensagem dizendo se é o mais alto ou o mais baixo.

25. Um avião voando em linha reta, a uma altitude **a,** passa sobre um ponto P situado em solo num instante t = 0. Se a velocidade **v** e a altura **a** forem lidos, calcular a distância **d** do avião ao ponto **P** após 1, 2, ... , 30 segundos. Escrever, para cada t, o tempo e a distância.

26. Escrever um algoritmo que lê um valor **n** que indica quantos valores devem ser lidos para **m,** valores todos inteiros e positivos, com leitura de um valor de cada vez. Escreva, para cada valor lido, o próprio valor lido, seu fatorial e sua raiz cúbica.

27. Escrever um algoritmo que gera e escreve os 5 primeiros números perfeitos. Um número perfeito é aquele que é igual a soma de seus divisores, exceto ele próprio.

    Exemplo: 6 = 1 + 2 + 3; 28 = 1 + 2 + 4 + 7 + 14

28. Escrever um algoritmo que lê 50 valores, um de cada vez, e encontra e escreve o maior deles.

29. Escrever um algoritmo lê **a, b, c, d** e **e** , que constituem o Gabarito de uma prova de 5 questões. Leia, a seguir, um número não determinado de conjuntos de 6 valores: **num, a1, a2, a3, a4 e a5**, onde **num** representa o número do aluno e os demais valores são as respostas daquele aluno às 5 questões da prova. Conte o número de acertos e multiplique por 2. Escrever, para cada aluno, o seu número e a sua nota.

30. Foi feita uma pesquisa em um município. A pesquisa coletou, para cada habitante, os seguintes dados: idade, sexo (M/F) e salário. Escreva um algoritmo que lê os conjuntos de dados obtidos, um conjunto de cada vez, obtém e escreve as seguintes informações:

    a) A média de salário dos habitantes do município.

    b) A maior e a menor idade das pessoas pesquisadas.

    c) A quantidade de mulheres com salário até 500,00.

    O algoritmo deve terminar quando for fornecida uma idade negativa.

31. Foi realizada uma pesquisa de algumas características físicas entre a população de uma determinada região. Os dados coletados por habitante, para serem analisados foram:

    - sexo (masculino ou feminino)

    - cor dos olhos (azuis, verdes ou castanhos)

- cor dos cabelos (louros, castanhos, pretos)

- idade

Escrever um algoritmo que obtenha e escreva:

- A maior idade entre os habitantes

- A quantidade de indivíduos do sexo feminino cuja idade está entre 18 e 24 anos inclusive e que tenham olhos verdes e cabelos louros

O final do algoritmo ocorrerá quando entrar uma idade negativa.

32. Escrever um algoritmo que lê uma quantidade não determinada de números positivos. Calcula e escreve a quantidade de números pares e ímpares, a média dos valores pares e dos ímpares e a média geral dos valores lidos. O algoritmo finaliza quando entrar um valor nulo.

33. Escrever um algoritmo que lê o código e o valor de cada produto de uma empresa. Supondo que a empresa deseja aumentar todos os produtos de códigos pares em 15% e todos os produtos cujos códigos são ímpares em 20%, escreva a lista dos produtos com o respectivo código e o seu preço já reajustado. O algoritmo termina quando entrar um código nulo.

34. Escrever um algoritmo que gera os números entre 1000 e 2000 e escreve aquele que divididos por 11 deixam resto igual a 5.

35. Escrever um algoritmo que lê 50 valores inteiros e positivos e:

    a) apresenta o maior dos 50 valores lidos;

    b) apresenta o menor dos 50 valores lidos;

    c) apresenta a média aritmética dos 50 valores lidos.

36. Escrever um algoritmo que lê um valor inteiro e positivo **n** e calcula a seguinte soma:

    $S = 1 + 1/2 + 1/3 + 1/4 + ... + 1/n$

    O algoritmo deve escrever cada um dos termos da soma e o valor final de S.

37. Escrever um algoritmo que calcula e escreve a média ponderada com peso 1.0 para os valores ímpares e peso 2.0 para os valores pares situados no intervalo [13, 77].

38. Escrever um algoritmo que lê 50 pares de valores, cada par formado pela altura e pelo sexo de uma pessoa (código=1, masculino e código=2, feminino) e escreve a maior e menor altura para cada sexo.

39. Escrever um algoritmo que calcula e escreve o produto dos números primos entre 95 e 1475.

40. Foi feita uma estatística nas 20 principais cidades do estado para coletar dados sobre acidentes de trânsito. Foram obtidos os seguintes dados:

    - código da cidade (1 a 20)

    - número de veículos de passeio do ano passado

    - número de acidentes com vítimas do ano passado

    Escrever um algoritmo que lê 20 conjuntos de valores, um de cada vez, contendo o código da cidade, o número de veículos de passeio e o número de acidentes e, calcula e escreve:

    a) Qual o maior e menor índice de acidentes e a que cidade pertencem respectivamente

    b) Qual a média de veículos nas cidades pesquisadas

c) Qual a média de acidentes entre as 20 cidades pesquisadas.

41. Fez-se uma pesquisa entre os 1200 habitantes de uma região para coletar os seguintes dados: sexo (0-feminino, 1-masculino), idade e altura.

    Escrever um algoritmo que lê as informações coletadas e escreve as seguintes informações:

    a) média da idade do grupo;

    b) média da altura das mulheres;

    c) média da idade dos homens;

    d) percentual de pessoas com idade entre 21 e 53 anos.

42. Escrever um algoritmo que lê um conjunto não conhecido de pares de valores **a** e **b**, todos inteiros e positivos, e, para cada par de valores lido, obtém o M.D.C. de **a** e **b** e o escreve juntamente com os valores lidos.

43. Escrever um algoritmo que lê um conjunto não conhecido de pares de valores **a** e **b**, todos inteiros e positivos, e, para cada par de valores lido, obtém o M. D. C. e o M. M. C. de **a** e **b** e os escreve juntamente com os valores lidos.

44. Escrever um algoritmo que lê **n**, inteiro e positivo, e calcula e escreve o termo de ordem **n** da sucessão abaixo:

Ordem:	1	2	3	4	5	6	7	8	9	10
Sucessão:	-1	0	5	6	11	12	17	18	23	24

45. Foram entrevistadas 50 pessoas e coletados os seguintes dados:

    a) sexo: M (masculino) e F (feminino);

    b) cor dos olhos: A (azuis), V (verdes), e C (castanhos);

    c) cor dos cabelos: L (louros), C (castanhos) e P (pretos);

    d) idade.

    Escrever um algoritmo que lê os 50 conjuntos de valores para cada uma das pessoas entrevistadas e escreve:

    1. a maior idade do grupo;

    2. a quantidade de indivíduos do sexo feminino, com idade entre 18 e 24 anos e que tenham olhos verdes e cabelos louros.

46. Uma loja de departamentos que tem cadastrados 180 clientes deseja mandar uma correspondência aos melhores dos seus clientes anunciando-lhes um bônus especial. Escrever um algoritmo que lê o nome do cliente e o valor de suas compras no ano que passou e calcula um bônus de 15% para aqueles clientes cujo valor de compras excedeu a 200 reais. Escrever o nome dos clientes que são merecedores deste brinde.

47. Escreva um algoritmo que lê um conjunto não determinado de pares de datas e escreva a diferença entre elas em dias.

48. Sabemos pela Lei de Newton que a força de atração entre dois corpos é diretamente proporcional ao produto

    de suas massas e inversamente proporcional ao quadrado da distância entre eles.

    Escrever um algoritmo que lê um número não conhecido de conjuntos de 4 valores **m1** (massa do primeiro corpo), **m2** (massa do segundo corpo), **G** (constante de gravitação universal) e **d** (distância entre os corpos)

e calcula e escreve, para cada conjunto lido, a força de atração entre os dois corpos de massas **m1** e **m2** respectivamente.

49. Supondo que a população de uma cidade A seja da ordem de 90.000 habitantes com uma taxa anual de

$$f = \frac{G_{m1m2}}{d^2}$$

crescimento de 3.1% e que a população de uma cidade B seja da ordem de 200.000 habitantes com uma taxa anual de crescimento de 1.5%. Escrever um algoritmo que calcula quantos anos serão necessários para que a população da cidade A ultrapasse a população da cidade B, mantidas as taxas atuais de crescimento de cada uma.

50. Supondo dados os três primeiros números primos, respectivamente 1, 2, e 3, escrever um algoritmo que escreve os 20 primeiros número primos, incluindo os três já dados.

CAPÍTULO V

ALGORITMOS DE MANIPULAÇÃO DE MATRIZES

O capítulo V tem como objetivo a conceituação de **matriz** apresentar os seus usos e os principais algoritmos relacionados.

A idéia de fazer uso de conjuntos de variáveis vem desde os primórdios da matemática. O próprio conceito de matriz, que será usado em algoritmos em nada difere do conceito usado na matemática.

" Matriz é um conjunto de variáveis, cada uma podendo representar o valor de uma constante, como se fossem variáveis simples, mas todas elas compartilham o mesmo nome."

Índices são associados a este nome comum com a finalidade de permitir a individualização dos elementos do conjunto.

Por exemplo: Um conjunto **A** de 5 elementos é representado como:

$A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5$

Onde A é o nome de cada um dos 5 elementos e a identificação dos elementos individuais, que não pode mais ser feita só pelo nome, que é o mesmo para todos, passa a ser feita pelo nome do conjunto seguido de um índice. Assim A_1 representa o primeiro elemento do conjunto e A_5 representa o quinto e último elemento do conjunto.

Muitos dos problemas do dia a dia exigem um número significativo de variáveis. Na maioria das linguagens de programação não é difícil encontrar um grande número de variáveis para representar os valores que são necessários para a solução do problema, mas em todas elas torna-se difícil e trabalhoso tratar com um grande número de variáveis diferentes.

As operações, tais como leitura, escrita e qualquer outra operação que envolva todas elas exigiria uma nomeação individual de todas estas variáveis.

Além disso, uma quantidade de problemas que com o uso de variáveis simples ficaria, praticamente, limitado a um pequeno número de valores, com a utilização de matrizes podem ser resolvidos de forma simples, para números grandes de valores.

Para diferenciar as variáveis simples das variáveis do tipo matrizes, as linguagens de programação usam, cada uma à sua maneira, uma indicação de quais variáveis são do tipo matrizes, quantas dimensões possui cada uma e qual o tamanho de cada dimensão.

Na linguagem algorítmica usada adotar-se-à a seguinte instrução:

```
Nome_da _Variável ( dim1, dim2, ... , dimn ): Mat
```

Onde a sigla Mat indica que a variável apresentada é do tipo Matriz com **n** dimensões.

Exemplo: V (5), M (6,8): Mat

Indica que V é uma matriz com uma dimensão (um vetor) e o tamanho da dimensão é 5, que M é uma matriz com duas dimensões, respectivamente de 6 linhas e 8 colunas.

A representação destas matrizes seria

V_1	V_2	V_3	V_4	V_5

M(6,8)

A maioria das linguagens de programação permitem trabalhar com matrizes de mais de duas dimensões. Apenas algumas como o BASIC estão limitadas a apenas duas dimensões. Se observamos que cerca de 95% dos problemas podem ser resolvidos com matrizes de duas dimensões, esta limitação passa a ser pouco significativa.

Consideremos, a seguir, algumas operações mais freqüentes com matrizes, tais como: leitura de matrizes, escrita de matrizes, soma dos elementos de uma matriz. A representação destas operações é mostrada abaixo:

a) Leitura de um vetor V(5)

```
Para i = 1 até 5
        Ler ( V( i ) )
Fim_Para i
```

b) Escrita de uma matriz M(5,4)

```
Para i = 1 até 5
    Para j = 1 até 4
        Escrever ( M(i , j) )
    Fim_Para j
Fim_Para i
```

c) Soma dos elementos de um vetor X(100)

```
S = 0
Para i = 1 até 100
    S = S + X( i )
Fim_Para i
```

Uma observação importante é que numa matriz M (i, j), por convenção o primeiro índice representa a linha e o segundo índice representa a coluna, mas como primeiro ou como segundo índices podem ser usadas quaisquer variáveis inteiras. Não existe nenhum compromisso do i com linha ou do j com coluna.

Se tivermos várias operações a serem feitas sobre a mesma matriz em ocasiões diferentes podemos usar índices diferentes. O que deve ser cuidado é apenas a variação do índice a qual deve ficar dentro dos limites da dimensão que o índice representa.

Considere-se o seguinte problema: ***" Ler um vetor N com 80 elementos, onde cada elemento representa a nota de um aluno de uma turma de Vestibular. Calcular a média geral da turma e verificar quantos alunos ficaram acima da média, quantos ficaram abaixo da média e quantos ficaram na média, escrevendo a média geral, o número de notas acima da média, o número de notas abaixo da média e o número de notas iguais à média ".***

Algoritmo 5.1: Cálculo da Média Geral

```
Algoritmo Estatística
N( 80 ) : Mat de tipo inteiro
S, Mg  : real
i, j, C, B : inteiro
Início
    S = 0
    Para i = 1 até 80
        Início
            Ler ( N( i ) )
            S = S + N( i )
        Fim_Para i
    Mg = S/80
    C = 0
    Para j = 1 até 80
        Se N( j ) < Mg então C = C + 1
    Fim_Para j
    B = 80 – C
    Escrever  ( "Média Geral  =          ", Mg,
               " Acima da Média =        ", C,
               " Abaixo da Média =       " , B )
Fim
```

Note que, conforme observação acima, na instrução " Para j = 1 até 80" podería-se ter usado novamente como índice a variável i em lugar do j.

Dada a natureza do problema, contar quantos alunos ficaram na média, acima ou abaixo dela, ele seria difícil de ser solucionado se os valores lidos não tivessem sido guardados em um vetor. Todos os valores teriam que ser fornecidos novamente para o computador de forma que pudesse compará-los com a média.

Considere-se o seguinte problema: ***" Ler um vetor V de 6 elementos e escrevê-lo. Ordenar, em seguida, os elementos de V em ordem crescente e escrever o vetor V assim ordenado."***

Na solução deste problema têm-se uma quantidade grande de algoritmos que solucionam o problema, alguns deles bem simples mas pouco eficientes e outros bastante eficientes, porém já nem tão simples.

O problema da eficiência do algoritmo, naturalmente só é relevante quando se trata de conjuntos grandes de elementos. No caso de um vetor de 6 elementos todos os algoritmos praticamente levam o mesmo tempo para fazerem a ordenação.

Veremos aqui duas destas soluções, bastante simples, mas em geral não muito eficientes para conjuntos grandes.

O primeiro método consiste em percorrer o vetor, comparando-se cada elemento do vetor com o elemento imediatamente seguinte (V(i) com V(i+1)). Se o elemento comparado for menor ou igual ao elemento seguinte, compara-se o segundo valor com o próximo até esgotar o vetor. Se o elemento comparado for maior do que o imediatamente seguinte, troca-se entre si os dois elementos comparados e continua-se comparando o segundo valor com o imediatamente seguinte enquanto houver elementos no vetor. Teríamos então:

```
Para i = 1 até 5
     Se V(i) > V(i+1) então Início
                                   x = V(i)
                                   V(i) =  V(i+1)
                                   V(i+1) = x
                              Fim
Fim_Para i
```

Supondo a leitura de um vetor V(6) com os seguintes valores:

15	7	1	4	17	10

Executando-se o trecho de algoritmo acima sobre este vetor, resultaria o seguinte:

7	1	4	15	10	17

Como pode ser observado, o novo vetor ficou mais próximo de uma ordenação em ordem crescente, mas não está ordenado conforme se deseja. Isto indica que o processo deve ser repetido várias vezes até se conseguir o objetivo final. Uma segunda execução nos levaria à seguinte configuração do vetor:

1	4	7	10	15	17

Como se vê, este vetor, com apenas duas execuções do trecho de algoritmo acima já ficou ordenado. Outros vetores, entretanto, podem exigir uma quantidade maior de execuções do trecho acima até conseguirem deixar o vetor totalmente ordenado. No máximo o número de execuções necessárias para a ordenação de um vetor é de N-1, onde N é o número de elementos do vetor.

Podemos assim completar o trecho acima:

```
Para n = 1 até 5
    Início
        Para i = 1 até 5
            Se V(i) > V(i+1) então Início
                                      X = V(i)
                                      V(i) =  V(i+1)
                                      V(i+1) = X
                                  Fim
        Fim_Para i
Fim_Para n
```

Este trecho certamente garante a ordenação do vetor conforme desejado, contudo, na prática o número de vezes que se tem que passar pelo vetor é bem menor do que N-1, conforme pôde ser visto no exemplo do vetor V acima. Em lugar de cinco passagens foram necessárias apenas duas. Desta forma, uma solução melhor é a seguinte:

```
k = 0
Enquanto k = 0
    Início
        k = 1
        Para i = 1 até 5
            Se V(i) > V(i+1) então Início
                                    X = V(i)
                                    V(i) =  V(i+1)
                                    V(i+1) = X
                                    k = 0
                                Fim
        Fim_Para i
    Fim_enquanto
```

Com isto a versão 1 da ordenação de um vetor V de 6 elementos é a seguinte:

Algoritmo 5.2 Ordenação pelo método da Bolha

```
Algoritmo OrdenaCrescenteBolha
V ( 6 ) : Mat de tipo Inteiro
i, k, x : Inteiro
Início
    Para i = 1 até 6
        Início
            Ler ( V( i ) )
            Escrever  ( V(i ) )
        Fim_Para i
    k = 0
    Enquanto k = 0
        Início
            k = 1
            Para i = 1 até 5
                Se V(i) > V(i+1) então Início
                                    x = V(i)
                                    V(i) =  V(i+1)
                                    V(i+1) = x
                                    k = 0
                                Fim
            Fim_Para i
        Fim_enquanto
    Para i = 1 até 6
        Escrever ( V ( i ) )
    Fim_Para i
Fim
```

Assim, o algoritmo acima lê um vetor V de 6 elementos e o escreve tal qual foi lido. Em seguida, ordena o vetor V em ordem crescente, usando um dos muitos métodos de ordenação existentes, denominado método da bolha (Bubble Sort). Uma vez ordenado, o algoritmo escreve o vetor ordenado em ordem crescente.

Outro processo de ordenação de um conjunto, consiste em achar o maior elemento do conjunto e trocá-lo com a última posição. Em seguida acha-se o maior do conjunto restante e troca-se com a penúltima posição. Procede-se da mesma maneira até que o conjunto restante contenha apenas um elemento quando se dá por concluído o processo.

Supondo que se tenha o mesmo vetor usado anteriormente:

15	7	1	4	17*	10*
15*	7	1	4	10*	17
10*	7	1	4*	15	17
4	7*	1*	10	15	17
4*	1*	7	10	15	17
1	4	7	10	15	17

Os * indicam os elementos que foram trocados de uma linha para a seguinte. A solução do problema da ordenação por este processo nos leva ao seguinte algoritmo:

Algoritmo 5.3 Versão 2 do algoritmo de ordenação de um conjunto pelo método da troca do maior.

```
Algoritmo OrdenaCrescenteTrocaMaior
V(6) : Mat de tipo inteiro
i, n, m, p, aux, k : inteiro
Início
    Para i = 1 até 6
        Início
            Ler ( V( i ) )
            Escrever ( V( i ) )
        Fim_Para i
    n = 6
    Enquanto n > 1
        Início
             m = V( 1 )
             p = 1
             Para i =2 até 6
                  Se m < V( i ) então Início
                                          m = V(i)
                                          p  =  i
                                      Fim
             Fim_Para i
            aux = V( n )
            V( n ) = V( p )
            V( p ) = aux
             n = n – 1
        Fim_enquanto
    Para k = 1 até 6
        Escrever ( V( k ) )
    Fim_Para k
Fim
```

Quando se tratar de matrizes de duas dimensões, tem-se dois índices que devem ser controlados. Considere o seguinte problema: ***“ Ler uma tabela T(10, 20). Em seguida ler 15 valores, um de cada vez, e, para cada valor lido, verificar se este valor está ou não na tabela T, escrevendo-o com a mensagem adequada.”***

Desta forma temos o algoritmo 5.5 abaixo:

```
Algoritmo Pesquisa em Tabela
T(10,20) : Mat de tipo inteiro
n, i, j, a : inteiro
Início
    Para i = 1 até 10
        Para j = 1 até 20
            Ler  ( T( i,j ) )
        Fim_Para j
    Fim_Para i
    Para n = 1 até 15
        Início
            Ler ( a )
            Para i = 1 até 10
                Para j = 1 até 20
                    Se a = T(i, j) então Início
                                            i = 12
                                            j =  10
                                        Fim
                Fim_Para j
            Fim_Para i
            Se i = 11 então escrever ( a, " não está  na Tabela" )
                        senão escrever ( a, " está na Tabela "  )
    Fim_Para n
Fim
```

Outro problema que aparece com freqüência quando se trata de matrizes de duas dimensões é o de fazer-se somas de diferentes partes da matriz. Isto será ilustrado através do seguinte problema:

"Ler uma matriz M(5,5) e efetuar as seguintes somas:

a) Soma dos elementos da linha 3.

b) Soma dos elementos da coluna 5.

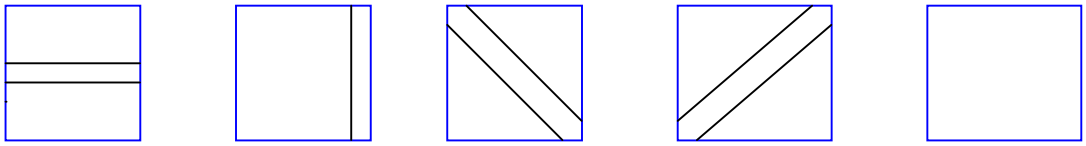
c) Soma dos elementos da Diagonal Principal.

d) Soma dos elementos da Diagonal Secundária.

e) Soma dos elementos de toda a matriz.

Escrever as somas calculadas."

As somas solicitadas podem ser melhor visualizadas abaixo:



Lembrando que qualquer variável pode representar o índice linha ou o índice coluna de qualquer matriz, não existindo qualquer compromisso do i com linha ou do j com coluna e resolvendo o problema por partes, temos o algoritmo abaixo:

Algoritmo 5.6 das Somas diversas de uma matriz (Primeira versão)

```
Algoritmo SomasMatriz
M(5,5) : Mat de tipo inteiro
i, j : inteiro
S1,S2,S3,S4,S5: real
Início
    Para i = 1 até 5
        Para j = 1 até 5
            Ler  ( M(i,j) )
        Fim_Para j
    Fim_Para i
    S1= S2 = S3 = S4 = S5 = 0
    Para i = 1 até 5
        S1 = S1 + M( 3,i  )
    Fim_Para i
    Para i = 1 até 5
        S2 = S2 + M( i, 5 )
    Fim_Para i
    Para i = 1 até 5
        S3 = S3 + M( i, i )
    Fim_Para i
    Para i = 1 até 5
        S4 = S4 + M( i, 5-i )
    Fim_Para i
    Para i = 1 até 5
        Para j = 1 até 5
            S5 = S5 + M( i, j )
        Fim_Para j
    Fim_Para i
   Escrever ( S1, S2, S3, S4, S5 )
Fim
```

Este algoritmo, quando se analisa o conjunto das ações que devem ser realizadas pode ser simplificado para o seguinte algoritmo:

Algoritmo 5.7 das somas elementares de uma matriz (Segunda versão):

```
Algoritmo SomasMatrizCompacto
M( 5,5 ): Matriz de tipo inteiro
i, j, S1, S2, S3, S4, S5: inteiro
Início
    / Leitura da Matriz /
    Para i = 1 até 5
        Para j = 1 até 5
            Ler  ( M( i, j ) )
        Fim_Para j
    Fim_Para i
    / Zerar os acumuladores das 5 somas /
    S1 = S2 = S3 = S4 = S5 = 0
    / Efetuar as somas a), b), c), d) /
    Para i = 1 até 5
    S1 = S1 + M ( 3, j )
    S2 = S2 + M ( i, 5 )
    S3 = S3 + M ( i, i )
    S4 = S4 + M ( i, 5-i )
    / Efetuar a soma de uma linha de M /
        Para j = 1 até 5
            S5 = S5 + M ( i, j )
        Fim_Para j
    Fim_Para i
    / Escrever as somas calculadas /
    Escrever ( S1, S2, S3, S4, S5 )
Fim
```

Principalmente usando matrizes, convém analisar o problema como um todo, antes de começar a construir o algoritmo, para evitar grande número de variações de índices o que pode tornar a solução extensa e pouco eficiente.

EXERCÍCIOS

01. Conceitue Matriz

02. Como é indicada a leitura de um vetor V(10)

03. Como se indica a escrita de uma matriz M (5, 8)

04. Escrever um algoritmo que lê um vetor V (6) e o escreve. Conte, a seguir quantos valores de V são negativos e escreva esta informação.

05. Escrever um algoritmo que lê um vetor A (10) e escreve todos os valores iguais a 5 deste vetor.

06. Escrever um algoritmo que lê um vetor X (100) e o escreve. Substitua, a seguir, todos os valores negativos de X por 1 e escreva o vetor modificado.

07. Escrever um algoritmo que lê um vetor V (10) e o escreve. Conte, a seguir o número de valores negativos de V, substitua os negativos por 1 e escreva a sua posição. No final escrever o número de valores negativos encontrados e o vetor modificado.

08. Escrever um algoritmo que lê um vetor C (50) e o escreve. Encontre, a seguir, o maior elemento de C e o escreva com a mensagem: È O MAIOR.

09. Escrever um algoritmo que lê um vetor N (80) e o escreve. Encontre, em seguida, o menor elemento e a sua posição e escreva estas informações.

10. Escrever um algoritmo que lê um vetor A (15) e o escreve. Ordene, a seguir, os elementos de A em ordem crescente e escreva o vetor A ordenado.

11. Escrever um algoritmo que lê um vetor N (20) e o escreve. Troque, a seguir, o 1° elemento com o último, o 2° com o penúltimo, e assim por diante, até o 10° com 11° e escreva o vetor N assim modificado.

12. Escrever um algoritmo que lê um vetor K (20) e o escreve. Troque, a seguir, os elementos de ordem ímpar com os de ordem par imediatamente seguintes e escreva o vetor assim modificado.

13. Escrever um algoritmo que lê um vetor M (20) e o escreve. Troque, a seguir, o 1° com o 11°, o 2° com 12° e assim por diante até o 10° com o 20° e escreva o vetor assim modificado.

14. Escrever um algoritmo que lê um vetor V (10) e um escalar. Fazer o produto do escalar A pelo vetor V e escrever o vetor e o produto calculado.

15. Escrever um algoritmo que lê um vetor G (20) que representa o gabarito de uma prova de 20 questões. Ler, a seguir, um número não determinado de conjuntos formados pelo número de um aluno e por um vetor R (20) que constitue o conjunto resposta do aluno àquela prova. Para cada aluno, calcular o número de acertos e depois calcular a sua nota pela fórmula:

$$\frac{\text{Número de acertos} - \dfrac{(20 - \text{número de acertos})}{2}}{2}$$

pois, cada dois erros descontam um acerto. Escrever o número do aluno, sua nota e a mensagem : APROVADO se tiver nota maior ou igual a 5 e REPROVADO se tiver nota menor do que 5.

16. Escrever um algoritmo que lê dois vetores N (10) e M (10) e faz o produto escalar de N por M escrevendo os vetores lidos e o produto calculado.

17. Escrever um algoritmo que lê dois vetores X (10) e Y (10) e os escreve. Crie, a seguir, um vetor Z que seja a união de X com Y. Escreva o vetor Z criado.

18. Escrever um algoritmo que lê dois vetores A (10) e B (10) e os escreve. Crie, a seguir, um vetor C que seja a interseção de A com B. Escreva o vetor C criado.

19. Escrever um algoritmo que lê dois vetores K (10) e N (10) e os escreve. Crie, a seguir, um vetor M que seja a diferença entre K e N. Escreva o vetor M criado.

20. Escrever um algoritmo que lê um vetor G (13) que é o gabarito de um teste da Loteria Esportiva, contendo os valores 1 (coluna 1), 2 (coluna 2) e 3 (coluna do meio). Ler, a seguir, para cada apostador, o número de seu cartão e o vetor resposta R (13). Verificar o número de acertos do apostador e escrever o número do cartão e o seu número de acertos. Se tiver 13 acertos, escrever também a mensagem : GANHADOR, PARABÉNS .

21. A empresa XYZ registra todas as suas despesa em um conjunto, à medida que os pagamentos vão sendo feitos. A receita da empresa, por sua vez, é realizada toda no dia 10 do mês e é registrada no mesmo vetor como um valor só. Como as despesas são registradas uma a uma e a receita de uma só vez, o valor da receita certamente será o maior valor do conjunto, mas não necessariamente será maior do que as somas das despesas. Encontre:

    a) O valor da receita;

    b) O valor das despesas;

    c) Diga se a empresa obteve lucro ou prejuízo.

22. Escrever um algoritmo que lê um vetor V (20) e o escreve. Compacte, a seguir, o vetor V, retirando dele todos os valores nulos ou negativos e escreva o vetor compactado.

23. Escrever um algoritmo que lê um vetor K (15) e o escreve. Crie, a seguir, um vetor N, que contém todos os valores primos de K e escreva o vetor N assim criado.

24. Uma agência de loteria esportiva deseja fazer uma estatística de números de acertadores por jogo em cada um dos testes. Escrever um algoritmo que lê o gabarito G (13) do teste em questão e em seguida lê os vetores resposta R (13) de cada apostador. No final deve escrever os jogos (1, 2 , 3 ... 13), o número de acertadores desta agência em cada jogo e o percentual de acertadores de cada jogo em relação ao número de apostadores.

25. Escrever um algoritmo que lê um vetor V (20) e o escreve. Retire, a seguir, os elementos em duplicata, compactando o vetor V e escrevendo o vetor compactado.

26. Escrever um algoritmo que lê um conjunto de 30 valores e os coloca em 2 vetores conforme forem pares ou ímpares. O tamanho dos vetores é de 5 posições cada. Se algum vetor estiver cheio escrevê-lo. Ao final escrever o conteúdo dos dois vetores. Cada vetor pode ser preenchido tantas vezes quanto for necessário.

27. Escrever um algoritmo que lê o 1º termo e a razão de uma progressão aritmética e gera os 20 termos seguintes desta progressão armazenando-os em dois vetores de 10 posições cada. No 1º vetor devem ser armazenados os termos cuja ordem de geração é ímpar e no 2º vetor, os termos cuja ordem de geração é par. Escrever, no final, os dois vetores de forma que os termos da progressão apareçam na ordem em que foram gerados.

28. Escrever um algoritmo que lê 2 vetores V (10) e X (10) e os escreve. Crie, a seguir, um vetor Y (20) com os elementos de V e X em ordem crescente (sem ter que ordenar o vetor Y) e escreva o vetor Y criado.

29. Escrever um algoritmo que lê para um vetor V (30), vinte valores que deverão ocupar as 20 primeira posições de V. Ordene, a seguir, os elementos de V em ordem crescente. Leia, em seguida, 10 valores, um por vez, e insira-os nas posições adequadas do vetor V, de forma que o mesmo permaneça ordenado em ordem crescente. Escreva o vetor V no final.

30. Escrever um algoritmo que gera os 10 primeiros números primos acima de 100 e os armazena em um vetor X(10) escrevendo o vetor X no final.

31. Escrever um algoritmo que lê um vetor L(5) que fornece os 5 números sorteados em algum sorteio da LOTO. Leia, a seguir, um número não determinado de vetores J (6) cada um contendo na primeira posição o número de identificação do cartão e nas demais posições as opções do apostador em questão. Para cada apostador escrever o número de seu cartão e o seu número de acertos, assinalando com uma mensagem quando tiver 3, 4 ou 5 acertos.

32. Escrever um algoritmo que lê um vetor X (20) e o escreve. Escreva, a seguir, todos os valores distintos que aparecem em X, com o seu número de ocorrências.

33. Escrever um algoritmo que lê uma matriz M (6, 6) e calcula as somas indicadas:

    a) Soma dos elementos acima da diagonal principal;

    b) Soma dos elementos acima da diagonal secundária;

    c) Soma dos elementos abaixo da diagonal principal;

    d) Soma dos elementos abaixo da diagonal secundária.

    Escrever as somas calculadas.

34. Escrever um algoritmo que lê uma matriz M (15, 15) e calcula as somas indicadas:

    a) Soma dos elementos simultaneamente acima das duas diagonais;

    b) Soma dos elementos simultaneamente abaixo das duas diagonais;

    c) Soma dos elementos que estão simultaneamente acima da diagonal principal e abaixo da diagonal secundária;

    d) Soma dos elementos que estão simultaneamente acima da diagonal secundária e abaixo da diagonal principal.

35. Escrever um algoritmo que lê uma matriz M (10, 10) e a escreve. Troque, a seguir, conforme indicação:

    a) A linha 2 com a linha 8;

    b) A coluna 4 com a coluna 10;

    c) A diagonal principal com a diagonal secundária;

    d) A linha 5 com a coluna 10;

    e) A linha 1 com a diagonal secundária;

    f) A coluna 7 com a diagonal principal.

    Escrever a matriz após cada troca.

36. Escrever um algoritmo que lê duas matrizes M (4, 6) e N (6, 8) e cria :

    a) Uma matriz M1 transposta da matriz M;

    b) Uma matriz M2 com todos os seus elementos iguais a 1;

    c) Uma matriz MI que seja uma matriz identidade;

    d) Uma matriz MP que seja o produto de M por N.

    Escrever as matrizes lidas e as matrizes criadas.

37. Escrever um algoritmo que lê uma matriz de coeficientes de um sistema linear e o vetor dos termos independentes do sistema e cria a matriz aumentada do sistema (n x n+1), escrevendo-a .

38. Escrever um algoritmo que lê duas matrizes N1 (4, 6) e N2 (4, 6) e cria :

    a) Uma matriz M1 que seja a soma de N1 com N2;

    b) Uma matriz M2 que seja a diferença entre N1 e N2.

    Escrever as matrizes lidas e criadas.

39. Escrever um algoritmo que lê uma matriz M(10, 20) e escreve:

    a) O maior elemento de cada linha da matriz;

    b) A média dos elementos de cada coluna;

    c) O produto de todos os elementos diferentes de zero;

    d) O número de elementos negativos da matriz M;

    e) A posição que é ocupada (linha e coluna) por um elemento cujo valor é lido.

40. Escrever um algoritmo que lê uma matriz M (6, 6) e um valor A e multiplica a matriz M pelo valor A e coloca os valores da matriz M multiplicados por A em um vetor V (36), escrevendo o vetor assim formado.

41. Escrever um algoritmo que lê uma matriz M (5, 5) e cria 2 vetores SL (5) e SC (5) que contenham, respectivamente as somas das linhas e das colunas de M. Escrever a matriz e os vetores criados.

42. Escrever um algoritmo que lê uma matriz M 50, 19), contendo nas posições assinaladas de cada linha o que segue:

    - coluna 1 → número do aluno;
    - colunas 2, 3 e 4 → notas da disciplina 1;
    - colunas 5, 6 e 7 → notas da disciplina 2;
    - colunas 8, 9 e 10 → notas da disciplina 3;
    - colunas 11, 12 e 13 → notas da disciplina 4;
    - colunas 14, 15 e 16 → notas da disciplina 5;
    - colunas 17, 18 e 19 → notas da disciplina 6.

    Calcular, para cada aluno, as médias de cada disciplina e a média geral armazenando num vetor o número do aluno , as médias por disciplina e a média geral e escrevendo o vetor antes de passar para o aluno seguinte.

43. Escrever um algoritmo que lê uma matriz A (12, 13) e divide cada um dos 13 elementos de cada uma das 12 linhas de A pelo valor do maior elemento em valor absoluto daquela linha. Escrever a matriz A lida e a matriz A modificada.

44. Escrever um algoritmo que lê uma matriz M(10, 10) e cria um vetor V1 contendo os elementos das linhas pares de M e um outro vetor V2 contendo os elementos da diagonal principal somados aos elementos que ficam na mesma linha, porém, na diagonal secundária. Escrever a matriz e os vetores criados.

45. Escrever um algoritmo que lê uma matriz M(6,6) e escreve os elementos que estão acima das duas diagonais simultaneamente.

46. Uma matriz esparsa é uma matriz que tem aproximadamente 2/3 de seus elementos iguais a zero. Escrever um algoritmo que lê uma matriz esparsa M(10, 10) e cria uma matriz condensada de 3 linhas, contendo os elementos não nulos de M, como segue:

    - a primeira linha contém a linha da matriz original onde foi localizado o elemento não nulo;
    - a segunda linha contém a coluna da matriz original onde foi localizado o elemento não nulo;
    - a terceira linha contém o elemento não nulo da matriz original.

    Escrever a matriz condensada criada e a matriz original lida.

47. Escrever um algoritmo que lê uma matriz A (8, 8), sem elementos repetidos e a escreve. Leia, a seguir, um número não determinado de valores para X, um de cada vez, e para cada X lido verifica se este valor está ou não está em A . Se for encontrado, encerrar a pesquisa escrevendo o valor X e a mensagem: “ESTÁ EM A”. Caso contrário, escrever X e a mensagem: “ NÃO ESTÁ EM A”.

48. Modificar o exercício anterior de forma a que, encontrando o valor, o algoritmo também escreva em que linha e coluna da matriz A ele foi encontrado.

49. Supondo que a matriz A do exercício acima possa conter elementos repetidos, modifique o algoritmo de sorte que ele escreva as várias posições em que o elemento X foi encontrado. Caso não exista nenhuma ocorrência escrever a mensagem: “ VALOR NÃO ENCONTRADO EM A”.

50. Em certos problemas de Cálculo Numérico é vantajoso que o elemento de maior valor em valor absoluto de uma coluna, numa matriz quadrada, esteja na diagonal, e a troca de linhas entre si não tem importância. Escrever um algoritmo que troque as linhas de uma matriz A (n x n) lida, de modo que:

    $| Dii | \geq | Aji |$ para i = 1 até n-1 e para j = i + 1 até n

    portanto, trocar a linha j com a linha i se $| A ji | > | Aii |$.

51. Muitos métodos de Cálculo Numérico para que possam ser aplicados exigem matrizes diagonalmente dominantes. Uma matriz é diagonalmente dominante se e somente se:

    $$(\forall i) \; | Aii | \geq \sum_{j \neq i} | Aij | \; e \; (\exists i) \; | \; | Aii | > \sum_{j \neq i} | Aij |$$

    Escrever um algoritmo que lê uma matriz M (4, 4) e verifica se esta matriz é diagonalmente dominante escrevendo uma mensagem adequada.

52. Na Teoria de Sistemas define-se como elemento **minimax** de uma matriz, o menor elemento da linha em que se encontra o maior elemento da matriz. Escrever um algoritmo que lê uma matriz K (10 , 10) e determina o elemento minimax desta matriz, escrevendo a matriz e a posição (linha e coluna) do elemento minimax.

53. Supondo que se lance um dado 50 vezes e cada vez o valor seja armazenado em um conjunto. Escrever um algoritmo que lê os valores dos lançamentos dos dados e:

    - determina e escreve o número de lançamentos nos quais o resultado obtido é maior do que a média aritmética dos 50 lançamentos;
    - determina a porcentagem de ocorrências da face 6.

54. Num determinado campeonato de futebol foram anotados os nomes dos jogadores e o número de gols que cada um marcou. Escrever um algoritmo que, à partir destas informações, forneça o nome do goleador do campeonato e o número de gols por ele marcados. Considere um campeonato com apenas 8 times e 12 jogadores cada.

55. Uma empresa vende 30 artigos. Cada artigo é identificado por um código e os artigos possuem preços variados. Usando variáveis indexadas escreva um algoritmo que lê os códigos dos artigos com respectivos preços e os armazena em matriz, determinando o seguinte:

    a) O código e o preço dos 3 artigos mais caros;
    b) A média dos preços dos 30 artigos;
    c) Quais os códigos dos artigos com preços superiores à média.

56. A empresa de Turismo XYZ TUR , na última temporada de verão fez uma pesquisa entre seus clientes no Rio Grande do Sul, para saber o seguinte: Praia de Preferência (Torres, Arroio do Sal, Curumin, Arroio Teixeira, Capão Novo, Capão da Canoa, Atlântida, Rainha do Mar, Mariluz, Imbé, Tramandaí, Cidreira, Pinhal e Quintão) e renda mensal. A pesquisa foi feita com 300 turistas e os dados foram armazenados em uma matriz com as linhas representando as praias na ordem apresentada e as colunas representando as seguintes faixas de renda (150-300, 301-500, 501- 800, 801 – 1000, 1001-2000, 2001- 4000 e 4001 – 10000).

    Escrever um algoritmo que lê estas informações e fornece os seguintes dados:

    a) Qual a praia preferida pelo maior número de turistas;
    b) A média da renda mensal dos turistas desta empresa;
    c) O número de turistas por praia;
    d) Renda média por cada praia.

57. Escrever um algoritmo que lê uma matriz T (10, 8) que contém os preços dos terrenos que são vendidos por uma imobiliária, de acordo com a metragem (colunas) e de acordo com a zona de localização (linhas).

    Ler, a seguir, um conjunto de valores, cada conjunto contendo o código do comprador, a zona de localização, a metragem do terreno e a opção de pagamento.

    Obter o preço do terreno e acrescentar mais 3% de corretagem.

    Se a opção de pagamento for 1(pagamento à vista) descontar 10% do valor total e escrever o código do comprador, a metragem do terreno, a zona de localização do terreno e o valor total a pagar.

    Se a opção de pagamento for 2 (Pagamento de uma entrada de 30% e mais 3 prestações de igual valor) então acrescentar 15% ao valor do terreno. Calcular a entrada e o valor de cada prestação. Escrever o código do comprador, a metragem, a zona de localização, a entrada a ser paga e o valor de cada prestação.

58. Uma matriz de adjacências, para um mapa rodoviário, é composta de elementos **zeros** e **uns** sendo que M(i, j) = 1 se existir uma ligação rodoviária da cidade i para a cidade j e M(i, j) = 0 se não existir esta ligação.

    Esta matriz será simétrica se para todo caminho i j existir um caminho j i (estradas de mão dupla).

    Uma matriz assim tem a propriedade de que M^2 representa os caminhos compostos de 2 trajetos, M^3 representa os caminhos compostos de 3 trajetos, e assim por diante.

    Se considerarmos apenas caminhos compostos que não passam duas vezes pela mesma cidade então os elementos da diagonal da matriz produto devem ser zerados.

Escrever um algoritmo que lê uma matriz de adjacências A para um conjunto de 10 cidades, e:

a) descubra quantos caminhos de 4 trajetos existem para cada par de cidades, escrevendo estas informações;

b) leia K e L, duas cidades quaisquer e verifique o número mínimo de trajetos para completar um caminho da cidade K para a cidade L, escrevendo esta informação.

59. Escrever um algoritmo que lê uma matriz M (15, 15) e a escreve. Verifique, a seguir, quais os elementos de M que estão repetidos e quantas vezes cada um deles aparece em M. Escrever cada elemento repetido com uma mensagem dizendo quantas vezes ele aparece.

60. Um quadrado mágico de ordem n (sendo n um número ímpar) é um arranjo de números de 1 até n^2 em uma matriz quadrada de tal modo que a soma de cada linha, coluna ou diagonal é a mesma.

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

A figura mostra um quadrado mágico de ordem 5. A regra de formação é relativamente fácil de ser verificada: Comece com o 1 no meio da primeira linha. À partir daí siga para cima e para à esquerda diagonalmente (quando sair do quadrado suponha que os lados superior e inferior estão unidos e os lados esquerdo e direito da mesma forma). Em cada quadrado que passar coloque o valor do quadrado anterior acrescido de uma unidade. Quando atingir um quadrado já preenchido, desça um quadrado e o preencha e continue seguindo a diagonal até ter colocado o valor n^2 .

Escrever um algoritmo que lê cinco números ímpares, um de cada vez, e gere e escreva o quadrado mágico correspondente aos números lidos.

61. Uma outra forma de representação de uma matriz esparsa é a matriz de 4 linhas. A matriz de 4 linhas Q é uma matriz com número de colunas igual ao número de elementos não nulos da matriz esparsa original.

Q (1, *) = valor dos elementos diferentes de 0 da matriz original.

Q (2, *) = linha da matriz esparsa onde aparece o elemento não nulo.

Q (3, *) = coluna da matriz esparsa onde aparece o elemento não nulo.

Q (4, *) = número que indica a coluna da matriz Q onde aparece o próximo elemento da mesma coluna.

Escrever um algoritmo que lê uma matriz A (15,15) que possui em torno de 2/3 de seus elementos nulos e cria e escreve a matriz Q de 4 linhas formada, segundo as regras acima.

CAPÍTULO VI

SUBALGORITMOS

Muitos problemas grandes ou nem tão grandes mas de solução mais complexa podem ser divididos, sucessivamente, em problemas menores, com lógica mais simples e de compreensão mais fácil. Em vez de escrever-se um algoritmo grande, escrevem-se vários algoritmos menores, os quais, não isoladamente mas em conjunto, resolvem o problema proposto.

Aos trechos de algoritmo que efetuam um ou mais cálculos determinados, dá-se o nome de **subalgoritmos.**

Os valores fornecidos aos subalgoritmos são denominados de **parâmetros.**

Ao processo de dividir-se problemas grandes em um conjunto de problemas menores denomina-se de **modularização**.

A modularização de um algoritmo tem como vantagens:

1. Dividir problemas grandes em vários problemas menores, de baixa complexidade, principalmente por terem um número pequeno de variáveis e poucos **caminhos de controle** (caminhos do início ao fim do programa).

   Isto resulta em maior qualidade do algoritmo uma vez que o controle de grande número de variáveis não é uma tarefa fácil, pois, sabe-se que as pessoas normais conseguem controlar 7± 2 variáveis simultâneas sem se atrapalharem. Com um número maior de variáveis a probabilidade de cometer equívocos aumenta. Além disso, o fato de os módulos, geralmente, terem poucos caminhos de controle simplifica a lógica e facilita o entendimento do que efetivamente deve ser feito. A qualidade de um algoritmo está fortemente associada à perfeita compreensão do algoritmo, pois, é muito mais provável que alguém produza um algoritmo correto se souber exatamente o que o algoritmo deve fazer.

2. A possibilidade de utilizar-se soluções gerais para classes de problemas em lugar de soluções específicas para problemas particulares.

   O uso de soluções genéricas, na maioria das linguagens de programação só é possível por meio do uso de módulos. O grande ganho que se tem com esta técnica é o alto grau de reusabilidade dos módulos escritos como soluções genéricas. Na prática, isto significa solucionar uma vez cada problema e não dezenas ou centenas de vezes.

3. Permite delimitar o escopo (nível de abrangência) de variáveis. As variáveis definidas no interior de um módulo são denominados de variáveis locais.

   As variáveis declaradas em um módulo são ativadas quando aquele módulo começa a ser executado. Ocupam memória somente até o final da execução do módulo ao qual pertencem. Isto permite uma otimização do uso da memória.

4. Evita a repetição, dentro de um mesmo algoritmo, de uma seqüência de ações em diferentes pontos.

   Escrevendo a seqüência de ações que se repete em vários pontos de um algoritmo na forma de um módulo, ganha-se tempo, encurta-se o algoritmo e dá-se menos chance ao azar de cometer algum erro em alguma destas seqüências.

Existem, na maioria das linguagens de programação, duas formas de implementação e uso de subalgoritmos (ou módulos):

a) Subalgoritmos que implementam uma função, no sentido matemático do termo, isto é, que calculam um único valor em função de um ou mais parâmetros recebidos;

b) Subalgoritmos do tipo procedimento que podem calcular um número qualquer de valores, calculados ou não em função dos parâmetros recebidos.

SUBALGORITMOS DO TIPO FUNÇÃO

A definição deste tipo de subalgoritmo será feita da seguinte forma:

```
Função NomeF (Arg1:tipo, Arg2:tipo, ..., Argn:tipo ): Tipo
   Definições de variáveis locais
   Início
      . . . . . . . . . . . . .
      . . . . . . . . . . . . .
      . . . . . . . . . . . . .
      NomeF = Expressão
      Retorne
   Fim
```

onde as linhas pontilhadas representam instruções comuns de um algoritmo, NomeF é o nome da função e Tipo é o tipo do valor que será devolvido pela função. O número de argumentos é qualquer e cada um vem acompanhado de seu tipo. Deve existir, no corpo do subalgoritmo, pelo menos uma instrução do tipo **NomeF = Expressão**. Como uma função só pode calcular um único valor, muitas linguagens de programação obrigam que este valor seja atribuído ao nome da função, dentro do corpo da função. Desta forma, como o nome da função é único, não tem como retornar mais de um valor, não existindo o perigo de alguém tentar descaracterizar o subalgoritmo como sendo do tipo função. A instrução Retorne indica o término da execução do subalgoritmo e o retorno do valor calculado para o algoritmo que chamou a função.

Quanto aos argumentos, que são os valores recebidos e/ou retornados por um subalgoritmo, podemos dividí-los em 3 categorias:

De Entrada : São os argumentos que têm seus valores estabelecidos fora do subalgoritmo e não podem ser modificados dentro do subalgoritmo (valores recebidos);

De Saída : São os argumentos que têm seus valores estabelecidos dentro do subalgoritmo (valores calculados);

De Entrada-Saída: São argumentos que têm seus valores estabelecidos fora do subalgoritmo, mas que podem ser modificados pelo subalgoritmo (valores modificados).

Quanto à passagem de parâmetros, que são os valores usados na chamada do subalgoritmo, existem duas formas, normalmente usadas:

Passagem por Valor : quando o que é passado para o subalgoritmo é o valor da variável usada como parâmetro;

Ex.: **Função F (x : inteiro, y : real): real**

onde x é um argumento do tipo inteiro passado por valor e y é um argumento do tipo real, também passado por valor.

Passagem por Referência: quando o que é passado não é o valor de uma variável, mas o endereço desta variável.

Ex.: **Função Func (ref Vet : vetor, dim : inteiro): inteiro**

onde Vet é um argumento do tipo vetor, passado por referência (isto é, pelo seu endereço) e dim é um argumento do tipo inteiro, passado por valor.

Todo algoritmo que referencia um ou mais subalgoritmos é denominado de **algoritmo principal** com relação aos subalgoritmos por ele referenciados. A referência a um subalgoritmo do tipo função dentro do algoritmo principal é feita em expressões aritméticas de forma semelhante ao uso de funções em contextos matemáticos.

Com relação aos subalgoritmos do tipo função, cabem as seguintes observações:

1. Um subalgoritmo do tipo função não é executável por si só. Necessita ser ativado por uma instrução de chamada de diferentes formas:

   a) Variável = NomeF (Parâmetros)

   b) Variável = ($X - Y^3$ + NomeF(Parâmetros))/ Z

   c) Escrever a+b, b^2-4ac, NomeF(Parâmetros), d-1

Como pode ser visto, a função se comporta como um operando em alguma expressão , em qualquer lugar onde podem aparecer expressões em um algoritmo.

2. Em subalgoritmos do tipo função, o valor calculado é retornado (na maioria das linguagens de programação) pelo nome da função. Isto implica na existência, dentro do corpo da função, de pelo menos uma instrução de atribuição de algum valor ao nome da função, sob pena de a função não retornar nada.
3. Os argumentos de definição do subalgoritmo são variáveis sem efeito, servindo apenas para se fazer a definição do subalgoritmo. Na hora da execução eles serão substituídos pelos parâmetros de chamada.
4. Os argumentos de definição e os parâmetros de chamada devem corresponder-se em número, tipo e na mesma ordem.
5. Enquanto os argumentos de definição devem ser obrigatoriamente variáveis, os parâmetros de chamada podem ser expressões .
6. Nos algoritmos será usada, como última instrução de uma função, explicitamente, a instrução Retorne. Em algumas linguagens, a simples atribuição de um valor ao nome da função eqüivale ao Retorne.
7. No caso de subalgoritmos do tipo função os argumentos são todos do tipo **Entrada**, isto é, devem ser estabelecidos fora da função e não podem ser modificados dentro da função. Apenas os seus valores podem ser usados na obtenção do valor a ser retornado pela função.
8. Normalmente, argumentos de Entrada em funções são recebidos por valor. Nos exemplos abaixo, todos os parâmetros de chama das funções apresentadas são por valor.

Exemplo de função com uso de variáveis simples:

"Seja escrever um algoritmo que lê um número não determinado de valores m, todos inteiros e positivos, um valor de cada vez, e, se m < 10 utiliza um subalgoritmo do tipo função que calcula o fatorial de m, e caso contrário, utiliza um subalgoritmo do tipo função para obter o número de divisores de m. Escrever cada m lido e seu fatorial ou seu número de divisores com uma mensagem adequada".

Neste caso, temos um programa principal e dois subalgoritmos. Vamos iniciar com a definição do Programa Principal.

Algoritmo 6.1:

```
Algoritmo Principal
x, m: Inteiro
Início
    Ler ( m )
    Enquanto m >0 e int(m) = m
        Início
            Se m < 10 então   Início
                                  x = Fatorial ( m )
                                  Escrever ( m, "fatorial = ", x )
                              Fim
                        senão  Início
                                  x =Numdiv ( m )
                                  Escrever ( m, " Numero de Divisores = ", x )
                              Fim
            Ler ( m )
        Fim_enquanto
Fim
```

Neste algoritmo aparecem as referências a dois subalgoritmos de nomes Fatorial e Numdiv, respectivamente, os quais devem ser escritos a seguir.

Algoritmo 6.2 : Subalgoritmo Fatorial:

```
Função Fatorial ( n: inteiro ) : inteiro
   Fat, i : inteiro
Início
   Fat = 1
   Para i = 2 até n
       Fat = Fat x i
   Fim_Para i
   Fatorial = Fat
   Retorne
Fim
```

Neste caso o argumento n é uma variável que serve apenas para se definir a função, sendo substituída pelo parâmetro m que é usado na chamada, isto é, a função calcula o fatorial de m fornecido como parâmetro de chamada e não de n usado como argumento de definição.

Algoritmo 6.3 : Subalgoritmo Numdiv:

```
Função Numdiv( n: inteiro ): inteiro
    i, ndiv : inteiro
Início
    ndiv =2
    Para i = 2 até m/2
        Se n  mod  i = 0 então ndiv = ndiv + 1
    Fim_Para i
    Numdiv = ndiv
    Retorne
Fim
```

Como pode ser visto, o valor calculado pela função é atribuído ao nome da função, antes do término da função. Como o nome da função é único, não há a possibilidade de se violar o conceito matemático de função, que calcula um único valor em função de seus argumentos.

Exemplo 2 de função com uso de vetor.

“ Escrever um algoritmo que lê um vetor V(10) e o escreve. Utilize uma função para obter o maior elemento de V. Escreva o maior elemento encontrado com a mensagem: “E o MAIOR do VETOR ”

Algoritmo Principal 6.4:

```
Algoritmo Principal
V(10 ): matriz de tipo inteiro
i, j, maior : inteiro
Início
    Para i = 1 até 10
        Início
            Ler ( V( i ) )
            Escrever ( V ( i ) )
        Fim_Para i
    maior = MaiorVetor ( V, 10 )
    Escrever ( maior, "E O MAIOR DO VETOR" )
Fim
```

O algoritmo principal chama a função MaiorVetor para obter o maior elemento do vetor V, a qual será apresentada, a seguir:

Algoritmo 6.5 : Função Maior

```
Função MaiorVetor ( Vet: vetor, m: inteiro ): inteiro
  i, mai : Inteiro
Início
    mai = Vet ( 1 )
    Para i = 2 até m
        Se mai < Vet ( i ) então mai = Vet ( i )
    Fim_Para i
    MaiorVetor = mai
    Retorne
Fim
```

Exemplo 3 de função com uso de matriz.

" Escrever um algoritmo que lê uma matriz M (6,6) e a escreve. Utilize uma função para obter a posição do menor elemento da diagonal principal de M. Escrever o menor elemento e sua posição na diagonal."

Algoritmo Principal 6. 6:

```
Algoritmo Principal
  M( 6,6 ): Matriz de tipo inteiro
  i, j, p :Inteiro
Início
    Para i = 1 até 6
        Para j = 1 até 6
            Início
                Ler ( M ( i, j ) )
                Escrever ( M( i, j ) )
            Fim_Para j
        Fim_Para i
    p = Posicao ( M , 6 )
    Escrever ( "MENOR = ", M( p,p), "E sua posição = ", p, p )
Fim
```

Este algoritmo faz uso da função Posicao para obter a posição do menor elemento da diagonal principal de uma matriz quadrada qualquer, que será apresentado a seguir:

Algoritmo 6.7 : Função Posição

```
Função Posicao( Mat: matriz, m: inteiro )
    i, j, pos, menor : Inteiro
Início
    menor = Mat( 1, 1 )
    pos = 1
    Para i = 2 até m
        Se menor > Mat( i, i ) então Início
                                        menor = Mat ( i, i )
                                        pos = i
                                    Fim
    Fim_Para i
    Posicao = pos
    Retorne
Fim
```

SUBALGORITMOS DO TIPO PROCEDIMENTO

É um tipo de subalgoritmo que pode calcular e retornar uma quantidade qualquer de valores. Neste caso, os valores calculados e que devem ser retornados figuram na lista dos argumentos de definição do subalgoritmo.

A definição deste tipo de subalgoritmo segue o padrão abaixo:

```
Procedimento NomeP( V1:tipo, V2: tipo, ...,Vn: tipo, R1: tipo, R2: tipo, ..., Rn: tipo )
        Definição de variáveis locais
    Início
        ......................................
        ......................................
        ......................................
        R1 = Exp1
        R2 = Exp2
        ................
        Rn = Expn
    Fim
```

Nos subalgoritmos que são procedimentos existem os três tipos de argumentos vistos anteriormente. Os argumentos V1, V2,..., Vn, que são valores recebidos pelo procedimento, (argumentos de Entrada), mas que podem ser argumentos que foram recebidos, modificados e retornados, (argumentos de Entrada-Saída), e os argumentos R1, R2, ... , Rn, que são valores calculados pelo procedimento e devolvidos, (argumentos de Saída).

Com relação aos subalgoritmos do tipo procedimento, cabem as seguintes observações:

1. Da mesma forma que a função, um subalgoritmo do tipo procedimento não é executável por si só. Necessita ser ativado por uma instrução de chamada da forma

   NomeP (Parâmetros)

2. Subalgoritmos do tipo procedimento podem calcular um número qualquer de valores, os quais, serão retornados através dos parâmetros de Saída ou de Entrada-Saída.
3. Da mesma forma que em funções, os argumentos de definição do procedimento são variáveis sem efeito, servindo apenas para se definir o procedimento. Na hora da chamada eles serão substituídos pelos parâmetros de chamada.
4. Como em funções, os argumentos de definição e os parâmetros de chamada devem corresponder-se em número, tipo e na mesma ordem.
5. Enquanto os argumentos de definição devem ser obrigatoriamente variáveis, os parâmetros de chamada podem ser expressões .

Exemplo1:

“ Seja escrever um algoritmo que lê um número não determinado de conjuntos de 3 valores a, b, c, não negativos, um conjunto de cada vez, e, para cada conjunto lido, calcula as médias aritmética, geométrica, harmônica e ponderada com peso 2 para o menor valor, 3.5 para o valor intermediário e 4.5 para o maior valor. Para cada conjunto lido escrever os valores lidos e as médias calculadas”.

Algoritmo Principal 6.8

```
Algoritmo Principal
  a, b, c, ma, mg, mh, mp: real
Início
    Ler ( a, b, c )
    Enquanto a x b x c ≠ 0
        Início
            Se a > 0 e b > 0 e c > 0
            então  Início
                        Ordena ( a, b, c )
                        Medias ( a, b, c, ma, mg, mh, mp, 2, 3.5, 4.5 )
                        Escrever ( a, b, c, ma, mg, mh, mp )
                    Fim
        Ler  (a, b, c )
        Fim_enquanto
Fim
```

onde aparece a referência (chamada) de dois procedimentos, Ordena que deverá ordenar os 3 valores em ordem crescente e Medias que calculará as 4 médias solicitadas.

Repare no subalgoritmo **Ordena** abaixo a solução para se ordenar em ordem crescente 3 valores, armazenados em 3 variáveis simples. Se este problema de ordenação de valores, que é um problema comum e freqüente no dia a dia, fosse tratado sempre com o uso de variáveis simples, verifique quantas instruções de seleção seriam necessárias para ordenar em ordem crescente 5 valores, por exemplo. Com isso você descobrirá facilmente porque este problema é resolvido preferencialmente com o auxílio de conjuntos (vetores). Isto simplifica a solução, tornando-a independente, em tamanho, do número de valores que devem ser ordenados.

No subalgoritmo **Ordena** todos os 3 argumentos são argumentos de **Entrada- Saída**, pois, os 3 valores x, y e z são recebidos pelo subalgoritmo em uma determinada ordem, e são devolvidos, possivelmente em uma outra ordem, isto é, são valores recebidos de um jeito, modificados, e devolvidos de outro jeito.

Já no subalgoritmo **Medias** existe uma nítida distinção entre os argumentos de **Entrada** (a, b, c, p1, p2, p3) que são os valores recebidos pelo subalgoritmo, e os argumentos de **Saída** (r1, r2, r3, r4), que são os valores calculados pelo subalgoritmo.

Algoritmo 6. 9 :Subalgoritmo Ordena:

```
Procedimento Ordena ( x, y, z:real  )
  aux : real
Início
    Se x > y então Início
                        aux = x
                        x = y
                        y = aux
                    Fim
    Se y > z então Início
                        aux = y
                        y = z
                        z = aux
                    Fim
    Se x > y então Início
                        aux = x
                        x = y
                        y = aux
                    Fim
Fim.
```

Algoritmo 6.10 : Subalgoritmo Medias :

Procedimento Medias (a, b, c, r1, r2, r3, r4, p1, p2, p3 : real)
Início

r1 = $\frac{a+b+c}{3}$

r2 = $\sqrt[3]{axbxc}$

r3 = $\frac{3}{1/a+1/b+1/c}$

r4 = $\frac{(axp1+bxp2+cxp3)}{p1+p2+p3}$

Fim.

Exemplo 2 com uso de vetor :

" Seja escrever um algoritmo que lê um vetor V(10) e o escreve. Ordene, a seguir, o vetor V em ordem crescente e escreva o vetor ordenado".

Algoritmo Principal 6.11

```
Algoritmo Principal
  V(10) : vetor de tipo inteiro
Início
    LeiaVet ( V, 10 )
    EscrevaVet ( V, 10 )
    OrdenaVet ( V, 10 )
    LeiaVet ( V, 10 )
Fim
```

Como pode ser visto, o algoritmo Principal não faz nada, mas manda outros fazerem o que tem que ser feito. Assim **LeiaVet**, lê um vetor, **EscrevaVet**, escreve um vetor e **OrdenaVet**, ordena um vetor. Embora o enunciado não explicite que se deva usar subalgoritmos para a leitura e para a escrita do vetor, isto é conveniente, pois, assim se resolve os problemas de leitura e escrita de vetores uma única vez, podendo reusá-los nos diferentes problemas que envolvem leitura e escrita de vetores.

Algoritmo 6.12 : Procedimento de Leitura de um Vetor:

```
Procedimento LeiaVet ( Vet: vetor, n: inteiro )
 i : inteiro
Início
    Para i = 1 até n
        Ler  (Vet ( i ) )
    Fim_Para i
Fim
```

Algoritmo 6.13 : Procedimento de Escrita de um Vetor:

```
Procedimento EscrevaVet ( Vet: vetor, n: inteiro )
 i : inteiro
Início
    Para i = 1 até n
        Escrever ( Vet ( i ) )
    Fim_Para i
Fim
```

Algoritm 6.14 : Procedimento de Ordenação de um Vetor:

```
Procedimento OrdenaVet ( Vet: vetor, n: inteiro )
 i, aux, k : inteiro
// Processo Método da Bolha //
Início
    k = 0
    Enquanto k = 0
        Início
            k = 1
            Para i = 1 até n-1
                Se Vet ( i ) > Vet ( i + 1 )
                então Início
                            aux = Vet ( i )
                            Vet ( i ) = Vet ( i + 1 )
                            Vet ( i + 1 ) = aux
                            k = 0
                        Fim
            Fim_Para i
        Fim_enquanto
Fim
```

Note-se que no subalgoritmo **LeiaVet** o argumento **Vet** é do tipo **Saída** e **n** é do tipo **Entrada**. Já no subalgoritmo **EscrevaVet** os dois argumentos **Vet** e **n** são do tipo **Entrada**. No subalgoritmo **OrdenaVet** o argumento **Vet** é de **Entrada-Saída** e **n** é só de **Entrada**.

Exemplo 3 com uso de Matriz:

"Seja escrever um algoritmo que lê uma matriz M(6, 8) e a escreve.

Utilize um subalgoritmo do tipo função para obter o maior elemento da linha 4 de M.

Escreva o valor do maior elemento da linha 4 de M.

Utilize um subalgoritmo do tipo procedimento para inverter os valores da linha 2 de M. Inverter, aqui, quer dizer, trocar o 1° com o último, o 2°, com o penúltimo, e assim por diante, até o 4° com o 5°.

Escreva a matriz modificada.

Utilize um subalgoritmo do tipo procedimento para trocar a linha 1 com a linha 5 de M.

Escreva a matriz modificada.

Utilize também procedimentos para a leitura e para a escrita da matriz".

Algoritmo Principal 6. 15

```
Algoritmo Principal
  M(6,6) : matriz de tipo inteiro
  Ma : inteiro
Início
    LeiaMat ( M, 6, 8 )
    EscrevaMat ( M, 6, 8 )
    ma = MaiorLinha ( M, 6, 8 , 4 )
    Escrever ( ma )
    InverteLinha ( M, 6, 8, 2 )
    EscrevaMat ( M, 6, 8 )
    TrocaLinhas ( M, 6, 8, 1, 5 )
    EscrevaMat ( M, 6, 8 )
Fim
```

Algoritmo 6.16 : Subalgoritmo LeiaMat que lê uma matriz de inteiros

```
Procedimento LeiaMat ( Mat: matriz, m, n: inteiro )
   i, j : inteiro
Início
    Para i = 1 até m
        Para j = 1 até n
            Ler ( Mat ( i, j ) )
        Fim_Para j
    Fim_Para i
Fim
```

Algoritmo 6.17 : Subalgoritmo EscrevaMat que escreve uma matriz de inteiros

```
Procedimento EscrevaMat ( Mat: matriz, m, n : inteiro )
   i, j : inteiro
Início
    Para i = 1 até m
        Início
            Para j= 1 até n
                Escrever ( Mat ( i, j ) )
            Fim_Para j
            Escrever
        Fim_Para i
Fim
```

Algoritm 6 . 18 : Subalgoritmo MaiorLinha que obtém o maior elemento de uma linha.

```
Função MaiorLinha ( Mat: matriz, m, n, l: inteiros ): inteiro
   i, maior : inteiro
Início
    maior = Mat ( l, 1 )
    Para i = 2 até n
         Se maior < Mat ( l, i ) então maior = Mat ( l, i )
    Fim_Para i
    MaiorLinha = maior
    Retorne
Fim
```

Algoritmo 6.19 :Subalgoritmo InverteLinha que inverte os elementos de uma linha de uma matriz.

```
Procedimento InverteLinha ( Mat: matriz, m, n, l: inteiro )
    aux, i : inteiro
Início
    Para i = 1 até n/2
        Início
            aux = Mat ( l, i )
            Mat ( l, i ) = Mat ( l, n+1-i )
            Mat ( l, n+1-i ) = aux
        Fim_Para i
Fim
```

Algoritmo 6.20 :Subalgoritmo TrocaLinhas que troca duas linhas quaisquer de uma matriz.

```
Procedimento TrocaLinhas ( Mat: matriz, m, n, l1, l2: inteiro )
    i, aux: inteiro
 Início
    Para i = 1 até n
        Início
            aux = Mat ( l1, i )
            Mat ( l1, i ) = Mat ( l2, i )
            Mat ( l2, i ) = aux
        Fim_Para i
 Fim
```

Observações:

1. Nos vários exemplos dados iniciou-se sempre com o Algoritmo Principal, passando-se em seguida para os subalgoritmos. Esta técnica denomina-se de TOP-DOWN. Suas principais vantagens são:

   a) Resulta em sistemas bem estruturados;

   b) Pode-se fazer testes de integração à medida que os módulos ficam prontos, sem prejuízo para os demais;

   c) Os módulos de controle, que são, geralmente, os que exigem maior cuidado, são os mais bem testados.

2. Na medida em que cada subproblema é resolvido na forma de um subalgoritmo escrito de forma genérica, isto é, para resolver uma classe de problemas e não um problema particular, ganha-se um alto potencial de reusabilidade destes módulos.
3. Diferentemente de funções, onde os argumentos são sempre argumentos de Entrada, em Procedimentos os argumentos podem ser de Entrada, Saída ou Entrada-Saída.
4. Nos exemplos apresentados até aqui, os nomes das variáveis usadas como argumentos foram diferentes das variáveis usadas como parâmetros, para caracterizar bem a diferença entre ambos. Contudo, nada impede que se utilize variáveis com mesmo nome. De qualquer forma, elas serão vistas pelo sistema como variáveis diferentes devido à sua abrangência. As variáveis definidas no subalgoritmo têm efeito apenas local. Com isso, podemos ter várias variáveis com mesmo nome, mas com significados diferentes em um mesmo sistema.
5. Nos exemplos apresentados, a passagem de parâmetros foi feita sempre por valor. Para evitar, em programas, que serão executados por um computador e que ocupam espaço na memória deste computador, que conjuntos de valores, principalmente grandes, passados como parâmetros, sejam duplicados quando do uso por algum subprograma, podemos passá-los não por valor, mas sim por referência, isto é, por endereço. Neste caso, o subprograma trabalha **com** ou **sobre** a mesma estrutura de dados do programa principal. Isto economiza memória e tempo de processamento para efetuar cópia da estrutura de dados.

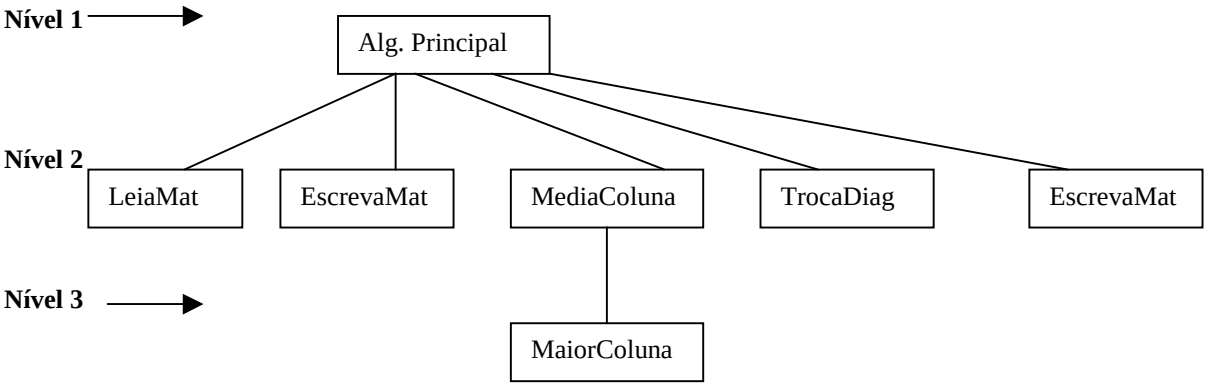
Exemplos de subalgoritmos com passagem de parâmetros por referência.

“ Seja escrever um algoritmo que lê uma matriz M (6, 6) e a escreve. Utilize uma função para obter a soma dos elementos da coluna 3de M, dividida pelo maior elemento desta coluna. Escreva o valor calculado. Utilize um procedimento para trocar entre si as duas diagonais de M e escreva a matriz assim modificada”.

Neste exemplo usar-se-á a passagem de parâmetros por referência em lugar de por valor. Para indicar que um parâmetro é passado por referência será usada a seguinte notação:

Ref Nome_da_Variável

Técnica TOP-DOWN



Nível 1

Algoritmo 6.21

```
Algoritmo Principal
  M(6,6) de tipo inteiro
  Media : inteiro
Início
    LeiaMat ( M, 6, 6 )
    EscrevaMat ( M , 6, 6 )
    Media = MediaColuna ( M, 6, 6, 3 )
    Escrever Media
    TrocaDiag ( M, 6 )
    EscrevaMat ( M, 6, 6 )
Fim
```

A técnica Top-Down inicia sempre pelo algoritmo principal e segue, de cima para baixo, e da esquerda para a direita. Portanto, seguem-se os subalgoritmos LeiaMat, EscrevaMat, MediaColuna, TrocaDiag, EscrevaMat e MaiorColuna nesta ordem.

Nível 2

Algoritmo 6.22 : Subalgoritmo LeiaMat

```
Procedimento LeiaMat ( ref Mat: matriz, m, n: inteiro )
   i, j : inteiro
Início
    Para i = 1 até m
        Para j = 1 até n
            Ler ( Mat ( i, j ) )
        Fim_Para j
    Fim_Para i
Fim
```

Agoritmo 6.23 : Subalgoritmo EscrevaMat

```
Procedimento EscrevaMat ( ref Mat: matriz, m, n: inteiro )
   i, j : inteiro
Início
    Para i = 1 até m
        Início
            Para j = 1 até n
                Escrever ( Mat ( i, j ) )
            Fim_Para j
            Escrever
        Fim_Para i
Fim
```

Subalgoritmo MediaColuna

```
Função MediaColuna ( ref Mat: matriz, m, n, c: inteiro ): inteiro
   i, soma, med, maicol: inteiro
Início
    maicol = MaiorColuna ( Mat, m, c )
    soma = 0
    Para i = 1 até m
        soma = soma + Mat ( i, c )
    Fim_Para i
    med = soma/maicol
    MediaColuna = med
    Retorne
Fim
```

Subalgoritmo TrocaDiag.

```
Procedimento TrocaDiag ( ref Mat: matriz, m: inteiro )
  i, aux : inteiro
Início
   Para i = 1 até m
       Início
           aux = Mat ( i, i )
           Mat ( i, i ) = Mat ( i, m+1-i )
           Mat ( i, m+1-i ) = aux
       Fim_Para i
Fim
```

O Procedimento EscrevaMat que é utilizado a seguir, já foi definido anteriormente e está sendo reutilizado já neste sistema, dispensando uma nova definição.

Nível 3

Subalgoritmo MaiorColuna.

```
Função MaiorColuna ( ref Mat: matriz, m, c: inteiro ): inteiro
   i, maior : inteiro
Início
    maior = Mat ( 1, c )
    Para i = 2 até m
        Se maior < Mat ( i, c ) então maior = Mat ( i, c )
    Fim_Para i
    MaiorColuna = maior
    Retorne
Fim
```

Existe uma outra técnica, freqüentemente usada na solução de problemas grandes, subdivididos em módulos (subalgoritmos), que é denominada de técnica **BOTTOM-UP.**

O uso desta técnica é aconselhado nos casos em que se conhece uma porção de detalhes do problema, mas não se tem ainda uma visão do todo da solução.

As vantagens desta técnica são:

1. Permitir paralelismo no desenvolvimento, diminuindo o tempo total do desenvolvimento da solução.

2. Permitir um melhor aproveitamento do pessoal disponível, pois, existirão módulos mais simples que podem ser desenvolvidos por quem tem menos experiência e, módulos mais complexos que devem ser desenvolvidos pelos mais experientes.
3. Como cada módulo é razoavelmente independente do resto do sistema, pode-se testá-lo em separado e além disso, fica mais fácil fazer-se um controle da padronização (conjunto de regras fixadas para o desenvolvimento de um módulo).

Como desvantagens desta técnica temos:

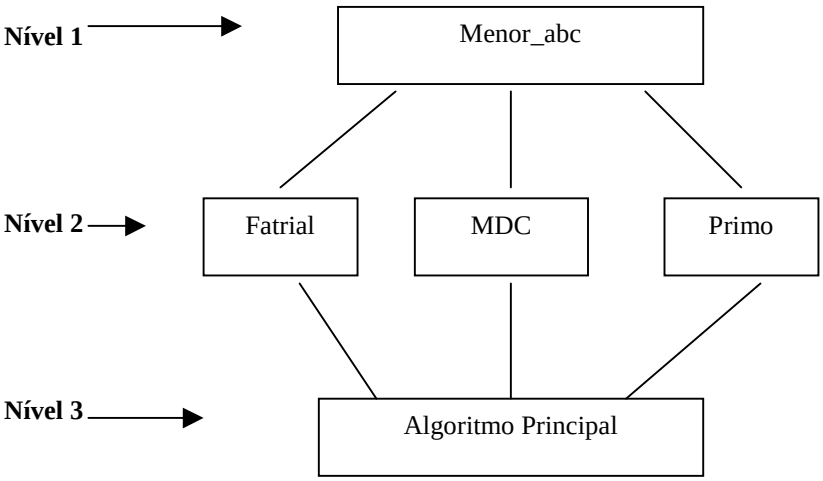
1. Se não houver um bom controle de padronização, o sistema como um todo pode resultar em uma **colcha de retalhos.**
2. A integração dos módulos no sistema não é feita, geralmente ao longo do período de desenvolvimento e sim no final do processo, podendo daí resultar um sistema potencialmente com mais falhas não descobertas durante o período do desenvolvimento.

Exemplo usando a Técnica Bottom-up.

" Seja escrever um algoritmo que lê um número não determinado de conjuntos de 4 valores i, a, b, c, todos inteiros e positivos, um conjunto de cada vez, e, para cada conjunto lido:

- **Se i = 1 então calcular o fatorial do menor entre os valores a, b, c, e escrever o menor valor e seu fatorial.**
- **Se i = 2 então calcular o MDC entre a soma dos dois maiores e o menor dos 3 valores e escrever a soma dos dois maiores, o menor e o MDC calculado.**
- **Se i = 3 então verificar se o menor dos 3 valores é primo ou não. Escrever o menor valor e uma mensagem dizendo se é primo ou não.**

Utilizar subalgoritmos para resolver os subproblemas envolvidos na solução do todo".



Nível 1

Como pode ser reparado na análise do que está sendo solicitado pelo algoritmo, em cada um dos 3 casos sempre é necessário primeiro conhecer-se qual o menor dos 3 valores a, b, c. Esta é a primeira ação que chama a atenção de quem lê o enunciado do problema.

Subalgoritmo Menor_abc

```
Função Menor_abc ( x, y, z : inteiro): inteiro
  Início
    menor = x
    Se menor  > y então menor = y
    Se menor > z  então menor = z
    Menor_abc = menor
    Retorne
Fim
```

Nível 2

Analisando-se um pouco mais o problema, aparecem três ações distintas, em função do menor dos 3 valores, o cálculo do fatorial do menor, a verificação de que o menor é ou não um número primo e o calculo do MDC também em função do menor.

Subalgoritmo Fatorial

```
Função Fatorial ( n: inteiro ): inteiro
   i, fat : inteiro
 Início
    fat = 1
    Para i = 2 até n
         Fat = fat x i
    Fim_Para i
    Fatorial = fat
    Retorne
Fim
```

Subalgoritmo MDC

```
Função MDC ( x, y: inteiro ): inteiro
   r: inteiro
 Início
    r = x mod y
    Enquanto r ≠ 0
         Início
              x = y
              y = r
              r = x mod y
         Fim_enquanto
    MDC = y
    Retorne
Fim
```

Subalgoritmo Primo.

```
Função Primo ( x: inteiro ): inteiro
    i : inteiro
Início
    Primo = 1
    Se x = 1 então Retorne
    Para i = 2 até √x
        Se mod ( x, i ) = 0 então Início
                                    Primo = 2
                                    Retorne
                                Fim
    Fim_Para i
    Retorne
Fim
```

Nível 3

Algoritmo Principal

```
Algoritmo Principal
  i, a , b, c, men, som, p, maxdiv: inteiro
 Início
     Ler ( i, a b, c )
     Enquanto i > 0 e i < 4
          Início
               men = Menor_abc( a, b, c )
               fat = Fatorial ( men )
               Escrever  ( men, fat )
               som = a + b + c – men
               maxdiv = MDC ( som, men )
               Escrever ( men, som, maxdiv )
               p = Primo ( men )
               Se p = 1 então escrever ( men, "E PRIMO" )
                         senão escrever ( men, " NÃO E PRIMO" )
              Ler ( i, a, b, c )
         Fim_enquanto
Fim
```

QUESTIONÁRIO

1. O que se entende por **modularização** ?
2. Quais as vantagens de se utilizar a modularização ?
3. O que se entende por **argumentos** em subalgoritmos ?
4. O que são **parâmetros** ?
5. Quais as diferenças mais significativas entre argumentos e parâmetros ?
6. Qual a diferença entre um parâmetro passado por **valor** e um parâmetro passado por **referência** ?
7. Quais as principais diferenças entre **funções** e **procedimentos** ou subrotinas ?
8. Qual a diferença entre uma variável **global** e uma variável **local** ?
9. Quais os 3 tipos de argumentos em um subalgoritmo ?
10. Qual a diferença entre a chamada de uma função e a chamada de um procedimento ?

EXERCÍCIOS

1. Escrever um algoritmo que lê dois vetores X(10) e V(15) e os escreve. Ordene, a seguir, os vetores X e V em ordem crescente e os escreva novamente. Crie, em seguida um vetor Y(25) contendo todos os valores pares de X e V. Escreva o vetor assim formado. Ordene o vetor Y em ordem decrescente e o escreva novamente. Utilize os subalgoritmos de leitura, escrita e ordenação em ordem crescente já apresentados anteriormente e escreva os restantes subalgoritmos necessários para a solução do problema. Utilize a técnica TOP-DOWN.
2. Escreva um procedimento que recebe um vetor Vet , sua dimensão e o tipo de ordenação que se quer (ordem crescente ou descrescente) e devolve o vetor ordenado de acordo com o desejado.
3. Escrever um subalgoritmo que recebe um vetor X, seu número de elementos e um parâmetro P que indica a forma de criação de um vetor Y à partir de X. Se P = 1 então criar Y com os elementos pares de X. Se P = 2 então criar Y com os elementos ímpares de X. Se p > 2 ou P < 0 então criar Y com todos os elementos nulos. Devolver o vetor Y criado.
4. Escrever um algoritmo que lê um vetor V (12) e o escreve. Encontre, a seguir, o menor elemento entre V(1), V(2) e V(3) e o escreva. Verifique, em seguida, se os valores V(4), V(5) e V(6) formam ou não um triângulo. Se formarem triângulo calcular e escrever a área deste triângulo. Caso contrário escrever " Os valores ", V(4), V(5), V(6), "não formam triângulo". Calcular, depois, a média ponderada de V(7), V(8) e V(9) com pesos respectivos de 4.0 para o maior dos 3 valores e peso 3.0 para os outros dois. Escreva a média com a mensagem "APROVADO" ou "REPROVADO" conforme a média calculada seja maior ou igual a 5.0 ou menor do que 5.0. Para finalizar, calcule o fatorial de cada um dos valores V(10), V(11) e V(12) e escreva os valores e seus fatoriais. Utilizar funções ou procedimentos já definidos ou por definir. Os subalgoritmos definidos especialmente para este problema devem ser apresentados. Utilize a técnica BOTTOM-UP.
5. O subalgoritmo abaixo é do tipo função, podendo, pois, calcular um único valor.

```
Função PRIMO ( x: inteiro ): inteiro
  x, prim,i : inteiro
Início
    prim = 1
    Se x ≠ 1 e x ≠ 2
        Então Para i = 2 até √x
                    Se x mod i = 0 então prim = 2
                Fim_Para i
    PRIMO = prim
    Retorne
Fim
```

Este subalgoritmo retorna 1 se x é primo e 2 se x não é primo. Modifique este subalgoritmo de forma que ele retorne, além da informação de que x é ou não primo, também o número de divisores de x, sem que deixe de ser do tipo função.

6. Escrever um subalgoritmo de nome TROCA que recebe como parâmetros x, y, z, e p, onde x, y, z são 3 valores reais quaisquer e p é um parâmetro que indica o que deve ser feito com os 3 valores x, y, e z. Se p = 1 então o procedimento deve trocar entre si os valores de x e y. Se p = 2 deve trocar entre si os valores de x e z. Se p = 3 deve trocar entre si os valores de y e z.

7. Escrever um algoritmo que lê um número ímpar entre 1 e 20 e utiliza um procedimento para gerar o quadrado mágico com a dimensão do número lido. Vide descrição do que é um quadrado mágico no exercício 60 do capítulo V.

8. Escrever um algoritmo que lê um valor A e um vetor V(30) e escreve o vetor lido. Conte, a seguir, quantos valores iguais a A estão em V e escreva esta informação. Utilize subalgoritmo do tipo função para obter esta informação. Crie, em seguida, um vetor X contendo todos os elementos de V que são diferentes de A, utilizando um subalgoritmo do tipo procedimento. Escreva o vetor X criado.

9. Dado o algoritmo abaixo, que efetua a soma dos elementos de uma matriz M (5, 5) que se encontram acima da diagonal secundária:

```
Função Soma ( ref Mat: matriz, m: inteiro ) : inteiro
  i, j, s : inteiro
Início
    s = 0
    Para i = 1 até 4
        Para j = 1 até 5-i
            s = s + Mat(i, j )
        Fim_Para j
    Fim_Para i
    Soma = s
    Retorne
Fim
```

Modifique esta função de modo que faça a soma dos elementos que estão acima da diagonal secundária de qualquer matriz com quaisquer dimensões, isto é, torne a função genérica.

10. Dado o algoritmo principal abaixo, escreva os subalgoritmos ORDENA1 e ORDENA2, sabendo que ORDENA1 utiliza o método da Bolha e ORDENA2, descobre o maior do conjunto e o troca com a última posição, encontra o maior dos restantes e o troca com a penúltima posição, e assim por diante.

```
Algoritmo Principal
  V(100) : vetor do tipo inteiro
 Início
    Ler m, n
    LeiaVet ( V, n )
    EscrevaVet ( V, n )
    Se n > 50 e m = 1 então ORDENA1( V, n )
                      senão ORDENA2 ( V, n )
    EscrevaVet ( V, n )
Fim
```

11. Uma matriz quadrada é dita simétrica se para todo i e j, o elemento (i, j) é sempre igual ao elemento (j, i). Escrever um subalgoritmo do tipo função que recebe uma matriz Mat e sua dimensão e verifica se esta matriz é ou não é simétrica.

12. O produto escalar de dois vetores A e B de comprimento n é dado por:

$$A \ . B \ \sum_{i=1}^{n} AiBi$$

Escrever um subalgoritmo do tipo função que recebe os vetores A e B e sua Dimensão n e retorna o produto escalar de A por B.

13. Escrever um subalgoritmo que recebe um vetor V que pode ter elementos em duplicata e altera o vetor recebido, substituindo os elementos repetidos por 0, retornando o vetor modificado e o número de modificações feitas. Substituir por 0 somente a segunda, terceira, etc, ocorrências de valores repetidos.

14. Escrever um algoritmo que lê dois vetores X e Y e chama o subalgoritmo do exercício 12 para calcular o produto escalar de X .Y. Se o produto escalar de dois vetores é zero, os vetores são ditos ortogonais. Se o valor retornado for diferente de zero, escrevê-lo com a mensagem " Produto escalar de X por Y". Caso contrário escrever a mensagem: "Os vetores X e Y são ortogonais".

15. Escrever um algoritmo que lê um número não identificado de valores inteiros de 5 dígitos cada um e utilize uma função para obter o dígito de controle deste número (dígito de controle = soma de cada um dos 5 dígitos multiplicados da esquerda para a direita respectivamente por 2, 3, 5 ,7,9 e dividido por 11). Utilize um procedimento que recebe o número lido e o devolve acrescido do dígito de controle (à direita) O programa deve escrever o valor lido e o valor acrescido de seu dígito de controle.

16. Escrever uma função que recebe um número para o qual o dígito mais da direita representa o dígito de controle, calculado da forma do exercício anterior, e devolve **verdadeiro** se o número é válido e **falso** se o número não confere.

17. Escrever um algoritmo que lê um número não identificado de valores n, todos inteiros e positivos, um por vez, e, para cada n lido chama uma função que verifica se n é um número perfeito ou não. Escrever o número lido e uma mensagem adequada.

18. Escrever um algoritmo que lê 50 conjuntos de 10 valores cada, contendo as seguintes informações:

    Na posição 1 : o número do aluno

    Na posição 2 : a nota da prova 1

    Na posição 3 : a nota da prova 2

    Na posição 4 : a nota da prova 3

    Na posição 5 : a média dos exercícios

    Na posição 6 : o peso da prova 1

    Na posição 7 : o peso da prova 2

    Na posição 8 : o peso da prova 3

    Na posição 9 : o peso dos exercícios

    Na posição 10: a freqüência do aluno

    Utilize uma função para obter a média geral do aluno.

    Se a freqüência < 75 então escrever o número do aluno, sua média geral e a mensagem "Reprovado por Falta de Freqência"

    Se a freqüência > 75 e media geral ≥ 7 então escrever o número do aluno, sua média geral e a mensagem "Aprovado por média".

    Se a freqüência > 75 e a media geral < 7 e > 4 então escrever o número do aluno, sua média geral e a mensagem "Aluno em Exame"

    Se a freqüência > 75 e média geral < 4 então escrever o número do aluno, sua média geral e a mensagem "Aluno reprovado por falta de aproveitamento".

19. Escrever um algoritmo que lê um número qualquer de valores n, todos inteiros e positivos, um valor por vez, e, para cada n lido utilize um procedimento para gerar e escrever a tabuada de 1 até 10 de n.

20. Escrever um algoritmo que lê dois vetores X e Y, cada um com 10 elementos e utiliza um procedimento para criar o vetor Z que seja a união de X com Y. Escrever o vetor união assim criado.

21. O mesmo que o exercício anterior, com a diferença de que o vetor Z criado deve ser a interseção de X com Y.

22. Escrever uma função que recebe um vetor e sua dimensão. O vetor contém um número binário, um dígito por posição. A função deve retornar o número decimal equivalente ao número binário do vetor.

23. Escrever um procedimento que recebe um número real positivo e devolve as suas partes, inteira e real em variáveis separadas.

24. Uma editora avalia a vantagem de editar ou não um livro, utilizando o custo de produção acrescido de 30%. Se este valor ficar abaixo de 40,00 para cada 100 páginas que o livro tiver, sua publicação é economicamente viável, caso contrário, não é viável. Construa um algoritmo que lê um conjunto não determinado de conjuntos de valores, cada conjunto constituído por: código de identificação do livro, custo fixo de um livro para a editora, número de exemplares estimado e número de páginas do livro. Para cada conjunto lido utilize uma função para obter o custo de produção:

    Custo de produção = custo fixo + número de exemplares x número de páginas x 0.030

    Utilizar um procedimento que verifica se a publicação do livro examinado é economicamente viável. Se for retorna um valor 1, caso contrário, retorna zero.

    Escrever os códigos dos livros cuja publicação tiver sido considerada econômica de acordo com os critérios da editora.

25. Escrever um procedimento que recebe duas matrizes: A(m,k) e B (k,n) e retorna a matriz produto C (m,n).

26. Goldbach fez uma conjetura (que não foi provada, mas exaustivamente testada sem encontrar contra-exemplo) de que qualquer número par maior do que 2 é a soma de dois números primos. Escreva um programa que gera os números pares entre 100 e 200 e utiliza um procedimento para encontrar os dois números primos que somados dão origem ao número examinado. Escrever o número par e os dois números primos cuja soma é igual ao número par escrito.

27. Escrever um procedimento que recebe o gabarito da LOTO e uma aposta (com no máximo 10 dezenas) e devolve o número de pontos da aposta.

28. Escrever uma função que recebe como parâmetros o dia, o mês e o ano e devolve o número de dias deste ano até a data fornecida. Lembre-se que Fevereiro pode ter 28 ou 29 dias conforme o ano for ou não um ano bissexto. Um ano bissexto é sempre divisível por 4.

29. Escreva um procedimento que recebe um número inteiro de no máximo 4 dígitos e devolve os dígitos separadamente em 4 variáveis.

30. Escreva uma função que recebe 4 valores, cada um de um dígito e devolve um número inteiro formado por estes 4 dígitos na ordem inversa em que foram fornecidos, desprezando zeros à esquerda.

31. Escrever um algoritmo que lê um número qualquer de valores n, todos inteiros e positivos com $n \leq 9999$ e utiliza o procedimento da questão 29 e a função da questão 30 para descobrir se n é um número capicua. Capicua é um número que lido da esquerda para a direita ou da direita para a esquerda é o mesmo. Escrever os números lidos que são capicuas.

32. Escrever um procedimento que recebe um valor inteiro n e devolve um vetor contendo os divisores de n , bem como uma variável contendo o número destes divisores.

33. Escrever um algoritmo que lê um valor n inteiro e gera valores inteiros de 1 até n. Para cada valor gerado utilize uma função para obter o número de divisores do valor gerado. Escrever uma tabela contendo os valores gerados e seu número de divisores, contando o 1 e o próprio número.

CAPÍTULO VII

ALGORITMOS RECURSIVOS

"Iterar é humano, a recursão é divina"

L. Peter Deutsch

Um objeto é dito recursivo se estiver definido em termos de si próprio. A recursividade é uma outra forma de se resolver problemas iterativos. A idéia que está por trás da recursividade é a velha estratégia de "dividir para conquistar", isto é:

- O problema é reduzido a um comjunto de problemas menores de mesma natureza e que, portanto, podem usar a mesma solução;
- Os problemas menores, se ainda não triviais, podem ser reduzidos a outros problemas menores ainda até que se tornem de solução trivial;
- As soluções dos vários subproblemas são combinadas dando origem, desta forma, à solução do problema original.

Isto implica na verificação de três condições:

1. Existe uma forma de se resolver o problema se ele for menor ?
2. É possível dividir-se o problema dado em subproblemas mais simples e de mesma natureza ?
3. A combinação das soluções dos subproblemas soluciona satisfatoriamente o problema original ?

Se estas três condições se verificam pode-se resolver o problema usando recursividade.

As soluções recursivas são, geralmente, mais concisas do que as suas equivalentes iterativas, mas exigem que se conheça todos os elementos importantes deste tipo de solução, a saber :

1. **Condição de Término.** O subalgoritmo recursivo inicia sempre com um teste que verifica se já foi alcançado o subproblema mais simples, cuja solução é trivial. Se este for o caso, o subproblema trivial é solucionado e o subalgoritmo termina a sua execução.
2. **Divisão.** Sempre que o problema atual não for ainda um problema tão simples que tenha uma solução trivial, divide-se o problema em subproblemas mais simples e de mesma natureza. Isto é feito por meio de uma chamada recursiva com os parâmetros do problema atual.
3. **Combinação.** Os resultados dos diferentes subproblemas são combinados, podendo envolver o uso de operações de adição, multiplicação ou outras, para a obtenção da solução do problema original.

A recursividade pode ser encontrada, não apenas na matemática, mas em muitas ações do dia a dia. Alguns exemplos são:

1. Andar

(a) Dar um passo;

(b) Dar um passo e Andar o resto do Caminho.

2. **Dizer um Discurso**

(a) Dizer uma frase;

(b) Dizer uma frase e Dizer o resto do Discurso.

3. **Ler um Livro**

(a) Ler uma página;

(b) Ler uma página e Ler o resto do Livro.

4. **Examinar Lista**

(a) Examinar um elemento;

(b) Examinar um elemento e Examinar Lista restante.

5. **Descendente**

(a) Filho ou Filha;

(b) Descendente de Filho ou Filha.

6. **Lista**

(a) Nada;

(b) Elemento seguido de Lista.

7. **Números Naturais**

(a) 0 é um número Natural;

(b) O sucessor de um número Natural é um número Natural.

8. **Função Fatorial para Inteiros positivos**

(a) Fat (0) = 1;

(b) Fat (n) = n * Fat (n – 1).

Segundo a Teoria da Computabilidade, existe solução para um problema, se ele puder ser enunciado de forma recursiva. Esta é, pois, a maneira natural (não necessariamente a mais eficiente) de se formular um problema e o modo mais simples de se entender os mecanismos de sua solução. O uso da recursividade se relaciona, em geral, com uma **generalização** do problema, no sentido de controlar mais do que apenas o resultado que se espera obter, mas também o mecanismo de obtenção deste resultado.

O poder da recursão deve-se ao fato de se conseguir, por seu intermédio, definir conjuntos infinitos por meio de formulações finitas. O uso da recursividade é particularmente recomendado quando a função ou o procedimento ou ainda a estrutura de dados a serem processados já estão definidos de forma recursiva.

Um procedimento **P** pode ser **diretamente** recursivo quando este procedimento contém uma referência explícita a si mesmo, ou, **indiretamente** recursivo quando P referencia um procedimento Q que por sua vez contém uma referência direta ou indireta a P. Logo, a simples inspeção do código pode dificultar a identificação de um procedimento recursivo.

Da mesma forma que em procedimentos repetitivos, nos procedimentos recursivos existe o problema da terminação dos cálculos, que deve ser cuidado.

Nas aplicações práticas com uso de recursão é importante mostrar que o nível mais profundo de recursão seja não apenas finito, mas bem finito. Caso contrário, o algoritmo, além de ficar pouco eficiente pode ainda estourar o espaço de memória, pois, cada nova chamada ativa novas variáveis locais e além disso, o estado do processamento também deve ser guardado para garantir um retorno correto após a chamada.

Considere-se a soma dos n números inteiros positivos:

Soma = 1 + 2 + 3 + . . . + n-2 + n-1 + n

ou

Soma(n) = n + n-1 + n-2 + ... + 3 + 2 + 1

A recursividade procura reduzir um problema qualquer a outro problema de solução mais fácil do que o anterior, de uma forma sucessiva até que o problema restante se torne um problema trivial.

1. Soma (n) = n + Soma (n-1)
2. Soma (n) = n + n-1 + Soma (n-2)

•

•

•

n-1 Soma (n) = n + n-1 + n-2 + ... + 3 + 2 + Soma (1)

n Soma (n) = n + n-1 + n-2 + ... + 3 + 2 + 1

No passo 1 o problema consistia da soma de n números inteiros. No passo 2 o problema já se resume a uma operação elementar de soma do valor n com a soma dos n-1 inteiros restantes, e assim sucessivamente até o passo n-1 onde se tem a soma de n + n-1 + n-2 + ... + 3 + 2 com a Soma de 1 inteiro restante.

Colocando-se este problema na forma de definição de problemas recursivos usada acima, tem-se:

Soma dos n inteiros

(a) Soma (n) = 1, se n = 1;

(b) Soma (n) = n + Soma (n-1), se n > 1.

Seja o algoritmo principal abaixo:

```
Algoritmo Principal
  n, s : inteiro
Início
    Ler ( n )
    s = Soma ( n )
    Escrever ( s )
Fim
```

Escrevendo-se a função Soma de forma recursiva:

```
Função Soma ( n: inteiro ): inteiro
Início
    Se n = 1 então Soma = 1
                senão Soma = n + Soma ( n-1 )
    Retorne
Fim
```

Para entender os algoritmos recursivos o principal problema é saber para onde deve ser retornado o resultado após a finalização de uma chamada recursiva. Supondo que o valor lido para n seja 5, seguem-se as seguintes chamadas e respectivos retornos:

Soma (5)

5 + Soma (4)

5 + 4 + Soma (3)

5 + 4 + 3 + Soma (2)

5 + 4 + 3 + 2 + Soma(1)

Os retornos acontecerão da seguinte forma:

Retorno 1 para a função → Soma = 1

Retorno 2 para a função → Soma = 1 + 2 = 3

Retorno 3 para a função → Soma = 3 + 3 = 6

Retorno 4 para a função → Soma = 6 + 4 = 10

Retorno 5 para o Alg. Princ. → Soma = 10 + 5 = 15

Outros exemplos:

Fatorial de n

(a) Fat (n) = 1 , se n = 0 ou n = 1

(b) Fat (n) = n x Fat (n – 1), se n > 1

Função que calcula o fatorial de n de forma recursiva:

```
Função FatRecursivo ( n:inteiro ) : inteiro
Início
   Se n = 0 ou n = 1 então FatRecursivo = 1
                     senão FatRecursivo = n x FatRecursivo ( n-1 )
   Retorne
Fim
```

Se em lugar de escrever uma função recursiva para cálculo do fatorial se quisesse um procedimento recursivo para cálculo do fatorial a solução seria:

```
Procedimento FatRecursivo ( n, fat : inteiro )
Início
   Se n = 0 ou n = 1 então fat = 1
                        senão Início
                                  FatRecursivo ( n-1, fat )
                                  Fat = fat x n
                              Fim
Fim
```

"Seja escrever um algoritmo que utiliza um procedimento recursivo para gerar o Triângulo de Pascal de ordem n, onde n é lido. Escrever o triângulo gerado."

```
Algoritmo Principal

T( 20,20 ) : matriz de tipo inteiro
i, j, n : inteiro
Início
   Ler ( n )
   TriangPascal ( n )
   Para i = 1 até n
       Para j = 1 até n
           Escrever ( T( i, j ) )
       Fim _Para j
       Escrever
   Fim_Para i
Fim
```

Procedimento recursivo que gera o Triângulo de Pascal:

```
Procedimento TriangPascal ( n : inteiro )
i : inteiro
Início
    T ( n, 1 ) = 1  || Primeira coluna é sempre 1 ||
    Se n > 1 então TriangPascal ( n-1 ) || Gera as n-1 primeiras linhas ||
    T ( n, n ) = 1        || O último da linha é sempre 1 ||
    Se n > 2 então Para i = 2 até n-1
                       T( n, i ) = T ( n-1, i-1 )  + T( n-1, i )
                   Fim_Para i
Fim
```

"Escrever uma função recursiva para calcular a raiz quadrada de X com uma aproximação inicial A e erro menor do que E, utilizando o método de Newton com X0 = A e, retornando o valor da raiz calculada"

```
Função RaizQuadrada ( X,A,E : real ): real
B : real
Início
  B = ( A x A + X ) / ( 2 x A )          || Calcula Nova Aproximação ||
   Se | B – A | < E                      ||  Verifica Limite de Erro      ||
       Então RaizQuadrada = B            ||  A  Aproximação está boa    ||
       Senão RaizQuadrada = RaizQuadrada ( X, B, E )
   Retorne
Fim
```

EXERCÍCIOS

1. Escrever um algoritmo que lê um número não determinado de valores m, todos inteiros e positivos e, de forma recursiva, verifica se o valor m lido é ou não primo, escrevendo-o com uma mensagem adequada.

2. Escrever uma função recursiva **Potência (X, Y)**, onde X e Y são argumentos inteiros e positivos, retornando X elevado a Y, usando a seguinte definição:

   (a) Potência (X, Y) = X, se Y = 1

   (b) Potência (X, Y) = X x Potência (X, Y –1)

3. Escrever uma função recursiva para obter o termo de ordem N da Série de Fibonacci (1,1,2,3,5,8,13,21,34,...), onde Fibonacci (N) é definido como:

   (a) Fibonacci (N) = 1 , se N = 1 ou N = 2;

   (b) Fibonacci (N) = Fibonacci (N-1) + Fibonacci (N-2), se N > 2.

4. Escrever um procedimento que recebe um vetor Vet por referência juntamente com sua dimensão n, e ordena, usando recursividade, o vetor Vet em ordem crescente. (Encontrar o índice do menor elemento e trocar o menor com o primeiro elemento, em seguida encontrar o menor do conjunto restante e trocá-lo com o segundo elemento, etc., até ordenar o vetor).

5. Escrever uma função inteira e recursiva **PesqBin(X, Vet, I, K)** para localizar X em um vetor Vet no intervalo Vet(I) a Vet (K) e retorna a posição do elemento X, se existir, ou retorna o negativo da posição I recebida.

6. Escrever uma função inteira e recursiva **PesqLin(X, V, K)** que localiza X no intervalo V (K) até V (N) por pesquisa linear, retornando a posição do primeiro elemento igual a X ou –1 se não encontrar X em V.

7. Escrever um procedimento recursivo **InverterVet(Vet, I, K)** que inverte o vetor Vet trocando entre si as posições Vet (I) e Vet (K), Vet (I + 1) e Vet (K –1), e assim por diante até completar a inversão.

8. Escrever uma função inteira recursisva **Menor(Vet, K, L)** que obtém o menor valor entre os K últimos elementos de Vet e o valor L fornecido como argumento, isto é, acha o mínimo entre Vet (K), Vet (K+1), ..., Vet (N) e L.

9. Escrever uma função recursiva real **Acumula(Vet, K)** que soma os K últimos elementos de Vet, isto é, soma os elementos de Vet (K) até Vet (N) retornando esta soma.

10. Escrever um procedimento recursivo **Transposta(Mat, K)**, onde Mat é uma matriz NxN e K indica à partir de que índice as sub-matrizes devem ser transpostas.

11. A função de Ackerman é definida da seguinte forma:

$$A(m, n) = \begin{cases} n + 1 & \text{se } m = 0 \\ A(n-1, 1) & \text{se } m \neq 0 \text{ e } n = 0 \\ A(m-1, A(m, n-1)) & \text{se } m \neq 0 \text{ e } n \neq 0 \end{cases}$$

   Utilize para o teste desta função valores pequenos para m e n.

12. Escrever um procedimento recursivo para ler um vetor V (20)

13. Escrever uma função recursiva para encontrar o maior elemento de um vetor V (30).

CAPÍTULO VIII

ALGORITMOS DE MANIPULAÇÃO DE CARACTERES

O tipo **caractere** consiste de um só dado desse tipo. Na maioria das vezes quando se trabalha com caracteres tem-se manipulações de dados formados por uma seqüência de caracteres, tais como: nomes, endereços, títulos de livros, frases ou textos.

Nas implementações de algoritmos de manipulação de caracteres nas diferentes linguagens de programação é óbvio que deve haver uma preocupação com a forma de armazenamento desse tipo de dados no computador. Não haverá, inicialmente, à nível de algoritmos este tipo de preocupação. Supõe-se aqui que a cada variável corresponda um espaço suficiente para guardar cada um de seus possíveis valores.

O tipo seq (seqüência) permite construir uma estrutura de dados cujos elementos são coleções de objetos (dados básicos), sem qualquer limitação no seu tamanho, exceto o de ser finito.

O formato de definição de uma seqüência é:

Seq : tipo

Logo, pode-se definir esse tipo de dado, denominado de **Cadeia de Caracteres** (String em Inglês) da seguinte forma:

Tipo Cadeia : seqüência de caracteres

Constantes desse tipo serão escritas da seguinte forma:

"a", "b", "valor", "Joao da Silva", " ", ""

onde o último exemplo representa uma cadeia vazia e o penúltimo representa uma cadeia formada unicamente pelo caractere branco.

Num tipo construído pelo mecanismo **seq** temos as seguintes operações primitivas que são comuns sobre cadeias de caracteres:

- **Concatenação** de duas ou mais cadeias : operação **concat**

  Ex.: X = "NACIONAL"

  "INTER" **concat** X ⟶ INTERNACIONAL

- **Primeiro** de uma cadeia de caracteres : operação **prim**

  Ex.: X = "INTERNACIONAL"

  Y = **prim** X ⟶ Y = "I"

- **Remover** o primeiro elemento da cadeia : operação **rem**

  Ex.: T = "VIRADA"

  T = **rem** T ⟶ T = "IRADA"

Estas operações serão consideradas operações primitivas, assim como o são, por exemplo, as operações "+" e "-" sobre dados do tipo inteiro.

Outras operações são interessantes e úteis quando se trabalha com cadeias de caracteres.

Eis alguns exemplos de operações construídas com o uso das operações primitivas apresentadas:

- **Contagem do número de caracteres de uma cadeia.**

```
Função Comprimento ( c: cadeia ) : inteiro
i : inteiro
c1 : cadeia
Início
    i = 0
    c1 = c
    Enquanto c1 ≠ “”   || Se c1 diferente de cadeia vazia  ||
        Início
            c1 = rem c1
            i = i + 1
        Fim_enquanto
    Comprimento = i
    Retorne
Fim
```

- **Obter uma Subcadeia à partir de uma cadeia c, iniciando na posição p e com comprimento m.**

```
Função Subcadeia ( c: cadeia, p, m : inteiro ) : cadeia
i : inteiro
c1, c2 : cadeia
Início
    Se m = 0 então Subcadeia = “”
                senão Início
                            c1 = c
                            c2 = “”
                            Se p > 1 então Para i = 1 até p-1
                                                c1 = rem c1
                                            Fim_Para i
                                            Para i = 1 até m
                                                Se c1 = “” então escrever “erro”
                                                            senão Início
                                                                        c2 = c2 concat prim c1
                                                                        c1 =  rem c1
                                                                    Fim
                                            Fim_Para i
   Retorne
Fim
```

Esta função elimina da cadeia c todos os caracteres anteriores à posição p que indica o início da subcadeia desejada. Em seguida, pega caractere por caractere até completar os m caracteres pretendidos e os concatena em c2. A variável c1 é usada para manter uma cópia de c enquanto a variável c2 é utilizada para armazenar a subcadeia que está sendo construída.

- **Obter a Posição inicial de uma subcadeia dentro de uma cadeia**

Embora também pudesse ser escrita na forma de uma função, esta operação será exemplificada através da construção de um procedimento.

O procedimento recebe uma cadeia c, uma cadeia s que se supõe seja uma subcadeia de c e retorna um parâmetro p contendo 0 caso a subcadeia s não seja encontrada em c ou retorna a posição inicial à partir da qual a subcadeia s foi encontrada em c se ela aí existir. Neste subalgoritmo também serão usadas variáveis auxiliares do tipo cadeia.

Assim a variável c1 será uma cópia de c. As variáveis c2 e c3 serão usadas durante a repetição para materem respectivamente cópias de c1 e de s já que c1 e s serão usadas para percorrer as respectivas cadeias durante a busca.

```
Procedimento PosiçãoSubCadeia( c, s : cadeia, p : inteiro )
i : inteiro
c1, c2, c3 : cadeia
Início
    Se s = "" então p = 0      || Situação em que s é vazio  ||
            senão Início
                    c1 = c
                    i = 1
                    Enquanto c1 ≠ ""
                        Início
                            Se prim c1 = prim s
                            então Início
                                    c2 = rem c1
                                    c3 = rem s
                                    Enquanto  c2 ≠ ""
                                        Início
                                            Se c3 = ""
                                            então p = i
                                            senão  Se prim c2 = prim c3
                                                    então Início
                                                            c2 =  rem c2
                                                            c3 =  rem c3
                                                        Fim
                                                    senão  Escape || Saída do enquanto mais interno  ||
                                        Fim_enquanto
                                    Se c3 = "" então p = i     || Sucesso  ||
                                Fim
                            C1 = rem c1
                            i = i + 1
                        Fim_enquanto
                    Fim
    p = 0  || Insucesso  ||
Fim
```

IMPLEMENTAÇÃO DE CADEIAS DE CARACTERES USANDO VETORES

Uma cadeia de caracteres muitas vezes é implementada como um vetor. Neste caso, cada posição do vetor pode conter um ou mais caracteres. Para se dimensionar adequadamente o vetor precisa-se saber o número n de caracteres da cadeia e o número m de caracteres que podem ser armazenados em cada posição do vetor. Assim np (número de posições do vetor) será definido como:

$$\mathbf{np} = \lfloor \mathbf{n/m} \rfloor \text{ (menor inteiro maior ou igual a n/m)}$$

No caso de se poder armazenar mais do que um caractere por posição há necessidade de uma operação que permita deslocar um caractere para a primeira ou a última posição dentro do espaço ocupado por cada elemento do vetor.

Como as linguagens de alto nível, normalmente não têm uma operação deste tipo, usar-se-á exemplos de vetores nos quais somente um caractere é armazenado em cada posição.

Em muitos casos, ao dimensionar o vetor não se tem conhecimento do comprimento de todas as cadeias que possam vir a ser armazenadas neste vetor. Por exemplo, se as cadeias a serem armazenadas forem nomes com tamanhos diferentes é interessante dimensionar o vetor de forma a comportar qualquer nome, independente de seu tamanho. Nestes casos, o final da cadeia pode ser indicado por um caractere especial, não pertencente à cadeia, como por exemplo: #.

Assim, se for armazenado o nome JOÃO DA SILVA num vetor V(16) tem-se:

J	O	Ã	O		D	A		S	I	L	V	A	#		

EXERCÍCIOS

1. Escrever uma função que recebe uma cadeia c e a devolve invertida.

   Ex.: Se c = “ACENOS” o procedimento deve retornar SONECA.

2. Escrever uma função **NumOcor** (cad : cadeia, c : caractere) que recebe uma cadeia cad e um caractere c e retorna o número de ocorrências do caractere c na cadeia cad.

   Ex.: Se cad = “INTERNACIONAL”

   então NumOcor (cad, I) ⟶ 2

   NumOcor (cad, N) ⟶ 3

3. Escrever um procedimento **UltimoCar** que recebe uma cadeia c e sem utilizar a função SubCadeia, devolve o último caractere da cadeia c.

   Ex.: Se c = “GREMIO”

   então UltimoCar (c) = “O”

4. Escrever um procedimento **MenosUltimo** que recebe uma cadeia c, e que não faça uso da função SubCadeia devolva a cadeia c sem o último caractere.

   Ex.: Se c = “IDENTIDADE”

   então MenosUltimo (c) = “IDENTIDAD”

5. Escrever um procedimento **RetiraSubCadeia** que recebe uma cadeia cad, a posição inicial P da subcadeia a ser retirada e N o comprimento desta subcadeia e devolve cad sem a subcadeia especificada.

   Ex.: Se cad = “ALGORITMOS”, P = 1 e N = 4 então RetiraSubCadeia (cad, P, N) = “RITMOS”

6. Escrever um procedimento **SubstitueCarac** que recebe duas cadeias cad1 e cad2, P um inteiro positivo que indica a posição inicial à partir da qual será feita a substituição de caracteres e N que indica o número de caracteres de cad1 que serão substituídos por caracteres de cad2.

   Ex.: Se cad1 = “INFORMAÇÃO” e,

   cad2 = “TERNO”

   P = 3

   N = 4

   então SubstitueCarac (cad1, cad2, P, N) = “INTERNAÇÃO”

   Se cad1 = “INFORMA” e cad2 = “ERNO”, P = 4 e N = 4 então SubstitueCarac (cad1, cad2, P, N) = “INFERNO”

7. Sabendo-se que num texto pode ocorrer o caractere “**,**” como sinal de pontuação ou como vírgula decimal entre dígitos numéricos. A vírgula será vírgula decimal somente nos casos em que for seguida imediatamente por um dígito. Escrever um procedimento para trocar as vírgulas decimais por “.” (ponto decimal) no texto abaixo, supostamente lido.

   “ José ganha 12,64 reais por hora. Como trabalha 40 horas semanais, se não tiver faltas, recebe um salário mensal de 505,60 reais”

8. Escrever um algoritmo que lê n (tamanho do nome), um nome de tamanho n para um vetor Nome (40) e um caractere a ser pesquisado. Os caracteres do nome são armazenados, um caractere por posição. Ao final da leitura deve ser colocado o caractere # que indica o fim do nome, se o mesmo não ocupar as 40 posições. O algoritmo deve usar uma função PesquisaVet que verifica se o caractere lido está ou não no vetor Nome.

9. Escrever um algoritmo que lê 10 conjuntos de valores, cada conjunto constituído de n (número de caracteres do nome a ser lido), do nome e de um sobrenome. Usando um procedimento deve verificar, para cada nome lido, se o sobrenome também lido pertence ou não a este nome. Se for encontrado deve retornar a posição inicial e final dentro do nome, os quais serão escritos pelo algoritmo principal. Caso contrário, deve retornar 0 para os dois valores e o algoritmo principal escreverá a mensagem adequada.

10. Escrever um algoritmo que lê para uma matriz 10 x 50 um texto. Em seguida lê 5 palavras, uma de cada vez, e, para cada palavra lida, conta quantas vezes ela aparece no texto, escrevendo a palavra lida e o seu número de ocorrências no texto.

CAPÍTULO IX

REGISTROS

Vetores, Matrizes e Conjuntos são estruturas de dados homogêneos, isto é, podem conter coleções de dados de mesmo tipo (todos inteiros, todos reais, todos caracteres). A diferença entre um vetor e um conjunto está apenas no fato de que um vetor pode ter dados repetidos e um conjunto não.

Na vida real encontramos muitas situações onde temos coleções de dados que são de tipos diferentes. Como exemplos temos: uma ficha de cadastro de um funcionário, uma cédula de identidade, uma inscrição em um concurso, etc.

Numa ficha de cadastro de um funcionário temos dados, tais como, nome do funcionário, endereço, número de dependentes, salário, função, entre outros.

Pode-se notar que estes diversos dados são de tipos diferentes. Assim, nome, endereço e função, são do tipo caractere, número de dependentes é do tipo inteiro e salário é do tipo real.

Para agrupar-se estes dados não servem os vetores, conjuntos ou matrizes. Há necessidade de um outro tipo de estrutura de dados que será denominado de registro.

Registro é, portanto, **uma coleção de dados heterogêneos,** isto é, de tipos diferentes.

A Declaração deste tipo de estrutura de dados será feita da seguinte forma:

```
Nome = Registro
          Campo 1: tipo
          Campo 2: tipo
           .
           .
           .
          Campo n: tipo
        Fim_Registro
```

Onde Nome é o nome do tipo de Registro que está sendo definido.

Campo 1, Campo 2, ..., Campo n representam os nomes de cada um dos campos do Registro.

Tipo representa qualquer tipo básico ou tipo anteriormente definido.

Portanto, a estrutura de dados representada pela ficha do funcionário pode ser representada por uma estrutura do tipo Registro, e sua definição é:

```
Ficha = Registro
          Nome(30)        : Caractere
          Endereço(20)    : Caractere
          Num_Dep         : Inteiro
          Salário         : Real
          Função(10)      : Caractere
        Fim_Registro
```

O exemplo mostra Ficha como sendo um modelo de um Registro ao qual está associada uma área de memória de nome Ficha, capaz de conter as 5 subdivisões constituídas por Nome, Endereço, Num_Dep, Salário e Função.

Neste caso, supondo que um tipo inteiro necessite de duas posições de memória, um tipo real de quatro posições e um tipo caractere de uma posição, a memória reservada para o Registro Ficha é de 66 posições.

A referência, por exemplo, ao nome do funcionário e ao número de dependentes, neste caso será feita da seguinte forma:

Ficha.Nome

Ficha.Num_Dep

Onde o nome do campo está ligado ao nome do Registro pelo caractere "." (ponto).

Suponha que a Ficha contenha mais um dado: Data de Admissão.

Neste caso a especificação da Ficha poderia ser:

```
Ficha = Registro
        Nome(30)       : Caractere
        Endereço(20)   : Caractere
        Num_Dep        : Inteiro
        Salário        : Real
        Data_Adm = Registro
                     Dia    : Inteiro
                     Mês    : Inteiro
                     Ano    : Inteiro
                   Fim_Registro
        Função(10)     : Caractere
      Fim_Registro
```

Neste caso, o campo Data é definido como uma nova estrutura do tipo Registro. Para referenciar o ano da data de admissão tem-se:

Ficha.Data.Ano

Uma das principais utilidades do uso de Registros é a referenciação genérica do conjunto de dados, como ocorre com vetores e matrizes.

Assim podemos ler ou escrever todos os dados da Ficha de uma só vez utilizando arquivos ou fazer outras referências genéricas :

Ler (Arq, Ficha)

Escrever (Arq, Ficha)

Cadastro = Ficha (que atribui a uma outra estrutura do tipo Registro a estrutura Ficha)

Registro de Vetores

Suponha que em um formulário de cadastramento de candidatos à uma eleição, as informações que devem ser preenchidas sejam as seguintes:

Nome do Candidato.

Endereço do Candidato.

Partido ao qual Pertence.

Cargo ao qual concorre.

Número de votos obtidos em eleições anteriores.

Este formulário poderia ser definido da seguinte forma:

Formulário = Registro

Nome(30) : Caractere

Ender(30) : Caractere

Partido(30) : Caractere

Cargo(10) : Caractere

Num_Votos(5) : Inteiro

Fim_Registro

Ou a definição poderia também ter sido feita desta outra forma:

Conjunto1 = vetor(30) de Caractere

Conjunto2 = vetor(10) de Caractere

Conjunto3 = vetor (5) de Inteiro

Formulário = Registro

Nome : Conjunto1

Ender : Conjunto1

Partido : Conjunto1

Cargo : Conjunto2

Num_Votos : Conjunto3

Fim_Registro

No caso do Número de votos, consideram-se apenas as 5 últimas eleições, de sorte que Num_votos(5) corresponde à última eleição, Num_Votos (4) à penúltima, e assim por diante. Se o candidato não participou de alguma eleição registra-se –1.

Como os campos do Registro são do tipo Vetor, diz-se que este Registro é um **Registro de Vetores.** Pode-se falar em Registro de Vetores mesmo que nem todos os campos do Registro sejam vetores.

A manipulação de Registros de Vetores obedece às regras de manipulação de cada uma das estruturas de dados utilizadas. Assim, para referenciar o partido do candidato usa-se:

Formulário.Partido

Para referenciar a quantidade de votos obtidos pelo candidato na antepenúltima eleição usa-se:

Formulário.Num_Votos(3)

Vetor de Registros

Voltando ao exemplo da empresa que utiliza uma ficha para registrar as informações sobre o funcionário. Como a empresa, normalmente, não tem apenas um funcionário, mas vários, também existirão várias fichas de funcionários, uma para cada funcionário. Isto nos leva a uma estrutura de dados composta, do tipo Vetor de Registros.

Na hipótese de uma empresa possuir 30 funcionários, cada um com a sua ficha de informações, esta situação poderia ser definida como segue:

Ficha = Registro

Nome (30) : Caractere

Ender (30) : Caractere

Num_Dep : Inteiro

Salário : Real

Função(10) : Caractere

Fim_Registro

Cadastro = Vetor (30) de Ficha

Neste exemplo, para poder definir Cadastro como um vetor de Ficha é necessário definir primeiro o tipo Ficha.

Para referenciar-se o nome do oitavo candidato cadastrado, a referência seria a seguinte:

Cadastro (8).Nome

Exemplo 1

"Construir um algoritmo, que, para cada rapaz identifica o grau de afinidade deste rapaz com cada uma das 5 candidatas que se apresentaram como interessadas. O algoritmo deve ler um conjunto não determinado de registros, cada um contendo: código (0 para rapaz e 1,2,3,4,5 para cada uma das 5 candidatas), nome do rapaz ou menina e vetor de respostas contendo as repostas às 25 questões (1 para SIM, 2 para NÃO e 3 para Indiferente). O término da leitura de registros se dará quando for lido um código –1.

A cada registro de rapaz seguem-se 5 registros de meninas. O algoritmo deve verificar para cada rapaz, qual a menina que tem mais afinidade com ele, escrevendo o nome do rapaz , o nome da menina e o grau de afinidade entre eles.

O grau de afinidade é obtido pelo número de respostas coincidentes às 25 perguntas. Para fins de seleção das candidatas o algoritmo escolherá a menina com número mais alto de respostas coincidentes. Em caso de empate no grau mais alto, deve escrever todos os nomes de meninas e respectivos graus".

O formato de cada Registro tanto de rapaz quanto de menina é o seguinte:

Código: _____

Nome: _____

Respostas:

Algoritmo Compatibilidade

C(5) : vetor de Inteiro

Conjunto = Registro

Código : Inteiro

Nome (30) : Caractere

Vet_Resp (25) : Inteiro

Fim_Registro

Conj(5) : vetor de Conjunto

```
Aux      : Conjunto
i, j, k : Inteiro
Início
    Ler ( Conjunto.Codigo )
    Ler ( Conjunto.Nome)
    Para i = 1 até 25
        Ler ( Conjunto.Vet_Resp(i) )
    Fim_Para i
    Aux = Conjunto
     Enquanto Conjunto. Código ≠ -1
          Para i = 1 até 5
               Ler ( Conj(i).Codigo )
               Ler ( Conj(i).Nome )
               Para i1 = 1 até 25
                    Ler ( Conj (i ).Vet_Resp( i1 ) )
                Fim_Para i1
               C ( i ) = 0
                    Para j = 1 até 25
                         Se Aux. Vet_Resp ( j ) = Conjunto. Vet_Resp ( j ) então C ( i ) = C ( i ) + 1
                     Fim_Para  j
          Fim_Para i
          Maior = C ( 1 )
          Para k = 2 até 5
               Se Maior < C ( k ) então Maior = C ( k )
          Fim_Para k
          Escrever ( Aux. Nome )
          Para k = 1 até 5
               Se Maior = C ( k ) então escrever ( Conj( k ). Nome, C ( k ) )
           Fim_Para k
           Ler ( Conjunto.Codigo )
           Ler ( Conjunto.Nome )
           Para k = 1 até 25
               Ler ( Conjunto.Vet_Resp ( k ) )
            Fim_Para k
           Aux = Conjunto
   Fim_ enquanto
Fim
```

Segue, abaixo, a versão deste algoritmo em Pascal usando subconjunto do Delphi :

```
{apptype = console }
program Compatibilidade;
uses
  Forms;
Type vetor=array[1..5]of integer;
     vetor1=array[1..25] of integer;
     vetor2=array[1..30] of String;
     Registro=Record
              Codigo   :Integer;
              Nome     :vetor2;
              Vet_Resp :vetor1;
            End;
vetor3= array[1..5] of Registro;
Var Aux,Conjunto : Registro;
    C            : vetor;
    i,j,k,Maior,i1  :Integer;
    Conj              : vetor3;
begin
  Readln ( Conjunto.Codigo);
  Readln ( Conjunto.Nome[30]);
  For i:= 1 to 25 do
    Readln (Conjunto.Vet_Resp[i]);
  Aux := Conjunto;
  While Conjunto.Codigo <> -1
  do Begin
       For i:=1 to 5 do
       Begin
         Readln ( Conj[i].Codigo);
         Readln ( Conj[i].Nome[30]);
         For i1:= 1 to 25 do
           Readln (Conj[i].Vet_Resp[i1]);
         C [ i ] :=0;
         For j:= 1 to 25 do
           if Aux.Vet_Resp[j] = Conj[i].Vet_Resp[j]
              then C[i]:=C[i] + 1 ;
       End;
       Maior:= C[1];
       For k:= 2 to 5 do
         if Maior < c[k] then Maior:= C[k];
       write ( Aux.Nome[30], '    '); writeln;
       For k:= 1 to 5 do
```

```
        if Maior = C[k] then write( Conj[k].Nome[30],'  ',C[k]);
      Readln ( Conjunto.Codigo);
      Readln ( Conjunto.Nome[30]);
      For i:= 1 to 25 do
        Readln (Conjunto.Vet_Resp[i]);
      Aux:= Conjunto;
    end;
end.
```

Segue uma versão do mesmo algoritmo em C

Compatibilidade.cpp

```
#include <stdio.h>
#include <conio.h>
void main()
  {
      struct Registro
        {
            int Codigo;
            char Nome[30];
            int Vet_Resp[25];
        } Aux, Conjunto, Conj[5];
      int C[5], i, j, k, i1, Maior;
      scanf("%d", &Conjunto.Codigo);  fflush(stdin);
      gets ( Conjunto.Nome );
      for ( i=0; i<25; i++)
      {
          fflush(stdin);
          scanf ( "%d", &Conjunto.Vet_Resp[i]);
      }
      Aux = Conjunto;
      While ( Conjunto.Codigo != -1 )
       {
           for ( i=0; i < 25; i++ )
           {
               scanf( "%d", &Conj[i].Codigo );
               fflush(stdin);
               gets ( Conj[i].Nome );
               for ( j = 0; j < 25; j++ )
                  {
                      fflush(stdin);
                      scanf( "%d", &Conj[i].Vet_Resp[j] );
                  }
```

```
            C[i] = 0;
            For ( i1 = 0; i1 < 25; i1++ )
                If ( Aux.Vet_Resp[ i1 ] == Conj[ i ]. Vet_Resp[ i1 ] ) C[i] = C[i] + 1;
        }
        Maior = C[0];
        For ( k = 1; k < 5; k++ )
            If ( Maior < C[k] ) Maior = C[k];
        Printf ("%s\n", Aux.Nome );
        For ( k = 0; k < 5; k++ )
            If ( Maior == C[k] ) printf ("%s     %d\n", Conj[k].Nome, C[k] );
        fflush(stdin);
        scanf ("%d", &Conjunto.Codigo );
        fflush( stdin );
        gets ( Conjunto.Nome );
        for ( i = 0; i < 25; i++ )
            {
                fflush( stdin );
                scanf ("%d", &Conjunto.Vet_Resp[i] );
            }
        Aux = Conjunto;
    }
}
```

EXERCÍCIOS

1. Considere um bilhete de Passagem de ônibus conforme mostrado abaixo:

VIAÇÃO OURO E PRATA

Número:_____ Nome da Empresa:_____

Origem:_____ Destino:_____

Data: _____/_____/_____ Horário: _____:_____

Assento: _____ Distância _____km

Valor da Passagem: _____

a) Defina uma estrutura de dados de nome Passagem.

b) Considerando um ônibus de 48 assentos, defina um Vetor de Passagens

2. Defina, baseado em seus conhecimentos, um registro contendo os dados de um Cheque Bancário.

3. A Biblioteca de Informática de uma Faculdade possui 2000 volumes, pertencentes a duas áreas: informática e administração. São 1200 títulos da área de informática e 800 títulos da área de administração. Como há previsão de expansão da biblioteca, respectivamente para 1500 títulos de informática e 1000 títulos de administração, os respectivos vetores já são definidos com estes tamanhos. Para a catalogação dos livros é utilizada uma ficha com as seguintes informações:

Código do Livro:_____ Área:_____

Título do Livro: _____

Autor: _____

Editora:_____ Ano:_____ Num. Exemplares:_____

a) Escrever um algoritmo que lê as informações necessárias para cada obra disponível na biblioteca reunindo-as em dois vetores: INF(1500) e ADM(1000).

b) Elaborar um procedimento que recebe os vetores INF e ADM, suas dimensões e um código de livro e escreve as informações referentes ao livro cujo código foi fornecido ou escreve uma mensagem dizendo que o código fornecido não foi encontrado.

c) Escrever uma função que recebe os vetores INF e ADM, suas dimensões e um código de livro e devolve 1 se o livro é da área de informática e 0 se for da área de administração.

d) Escrever uma função que recebe os vetores INF e ADM, suas dimensões e um código de livro e devolve o número de exemplares deste livro.

e) Escrever um procedimento que recebe os vetores INF e ADM, suas dimensões e um ano e lista todas as informações existentes sobre os livros editados neste ano.

f) Escrever um procedimento para inserir novos livros na biblioteca. O procedimento recebe os vetores INF e ADM, bem como as informações sobre o livro que deve ser inserido, alterando o respectivo vetor inserindo nele as informações deste novo livro. Se a inserção ocorreu com sucesso, escrever o nome do vetor, sua posição e as informações inseridas. Caso falte espaço, escrever a mensagem: “ Sem espaço para inserir as informações no vetor XXX”.

4. Suponha que o DETRAN queira informatizar o controle de veículos de uma dada cidade que possui um total de 6000 veículos. O registro das informações sobre os veículos é feito em uma ficha conforme modelo abaixo:

DETRAN – CADASTRO DE VEÍCULOS

Tipo de Veículo: _____ Marca: _____

Modelo: _____ Cor: _____ Chassi:_____

Proprietário: _____

Combustível: _____ Placa: _____ Ano: _____

Onde Tipo de Veículo: passeio, utilitário, caminhão ou ônibus;

Marca : Wolkswagen, Chevrolet, Ford, Fiat, Outro Nacional, Importado;

Combustível: gasolina, álcool ou diesel

Placa: 3 letras e quatro dígitos

Considere que estas informações já estão cadastradas em um vetor de Registros denominado CVA(6000).

a) Escreva um procedimento que recebe um ano de fabricação um tipo e um combustível e lista todos os proprietários cujos carros tenham ano de fabricação maior ou igual ao ano fornecido, que sejam do tipo igual ao que foi fornecido e que utilizem o mesmo combustível fornecido.

b) Escrever um procedimento que recebe uma letra e um dígito e lista todos os proprietários cujos veículos tenham como primeira letra da placa a letra fornecida e como último dígito da placa o dígito fornecido.

c) Escrever uma função que recebe uma marca de carro, uma cor de carro, as duas primeiras letras da placa e os dois primeiros dígitos da placa e verifica se existe algum carro com estas características, devolvendo o número do chassi.

d) Escrever um procedimento que recebe o número de um chassi e lista as restantes informações sobre aquele carro, se for encontrado algum carro com este número de chassi. Caso contrário, escreva a mensagem: “ Chassi inexistente”.

5. Uma Companhia de Turismo para fazer reservas de passagens entra com os dados cadastrais de cada passageiro conforme ficha abaixo:

COMPANHIA DE TURISMO SIRIUS LTDA

Nome: _____ Idade: _____

Endereço: _____

Vôo: _____ tipo: _____ Mês: _____

Preço Base: _____

O sistema fornece o valor a ser pago de acordo com as seguintes regras:

1. Se a idade do passageiro for maior ou igual a 65 ou menor do que 10 haverá 20% de desconto sobre o preço base;
2. Se o mês da viagem for de baixa temporada (março, abril, maio, junho, setembro ou outubro) há um desconto de mais 30% sobre o preço anterior.

Escrever um algoritmo que, para cada passageiro lido, escreve o nome do passageiro, seu vôo e o valor a ser pago.

6. Supondo que para o exercício anterior o pagamento seja feito por cheque, contendo as seguintes informações:

CHEQUE BANCO COOPERATIVA SA

Banco:_____ Agência: _____ Número do Cheque:_____

Número da Conta: _____DC: __ Valor: _____

Data: ___/___/___ Local: _____

Nome: _____

a) Construir um procedimento para a leitura destas informações.

b) Construir uma função para verificar a correção da conta, verificando o DC e retornando 0 ou 1 conforme a conta estiver ou não correta.

c) Incluir no algoritmo da questão 5 a chamada do procedimento e da função e acrescentar na escrita a mensagem: “pago com cheque:”,xxxxxx, “do Banco:”, xxxxx.

7. A CBF, como todos os anos vem fazendo, credenciou 500 jornalistas esportivos de todo o país, para fazer a eleição do melhor jogador e do melhor time do Campeonato Brasileiro do ano. Cada jornalista recebe a lista dos 24 times e dos 22 jogadores de cada time. Cada jornalista credenciado indica o que considera o melhor time e o melhor jogador do Campeonato no ano. Os votos de cada jornalista compreendem doi nomes, um de time e outro de jogador e serão lidos.

a) Apresente as estruturas de dados necessárias à solução deste problema.

b) Escreva um algoritmo que lê os dados (times e jogadores) e escreve uma lista que será entregue a cada jornalista. Em seguida, lê os votos de cada jornalista e emite um relatório com o resultado da eleição.

8. Escreva um algoritmo que soluciona o problema do preenchimento das vagas nos 5 cursos de um Centro Universitário. Cada candidato que prestou o Vestibular na instituição gerou um registro com os seguintes campos: número de inscrição, média geral obtida no Vestibular e código do curso para o qual se candidatou. Cada um dos 5 cursos oferecidos pelo Centro Universitário possui 50 vagas. O algoritmo deve ler os registros dos vestibulandos e distribuí-los pelos 5 cursos de acordo com a média obtida e do curso escolhido. Em caso de empate na média, será privilegiado o aluno com menor número de inscrição (o primeiro que se inscreveu) se este empate ocorrer na última vaga de um curso. O final de dados será indicado por um número de inscrição negativo.

9. A Agência Aproxima SA pretende implantar um sistema de controle de cadastro de seus clientes e descobrir afinidades entre os clientes. O algoritmo deve, basicamente, ter as seguintes operações:

a) Um procedimento CadastraNovoCliente que inclui os dados de um novo cliente. Não poderão existir dois clientes com mesmo nome.

b) Um procedimento AtualizaDadosCliente que altera dados de um cliente, exceto o seu nome.

c) Um procedimento RemoveCliente que retira um cliente do cadastro.

d) Um procedimento ListaCadastro que lista todas as informações de todos os clientes da empresa. Devem ser construídos pelo menos 3 outros procedimentos para se obter listagens específicas por critérios, tais como, sexo, idade, data de nascimento.

e) Um procedimento InformaDadosCadastrais que lista as informações de um determinado cliente.

f) Um procedimento ListaClientesCompatíveis que lista, para um determinado cliente, outros clientes que têm afinidades com ele.

Para cada cliente são cadastradas as seguintes informações: nome, telefone, sexo, idade, data de nascimento e interesses pessoais (Artes, Música, Tipos de Literatura, Teatro, Cinema, Esportes, Política, Lazer, Culinária, Turismo, Festas, Religião). Cada cliente deve informar de 3(mínimo) a 5 (máximo) áreas de interesse, numeradas na ordem de apresentação.

O critério para definir se duas pessoas são ou não compatíveis envolve:

- Sexos diferentes;
- Diferença de idade não superior a 10 anos;
- Possuir pelo menos 3 áreas de interesse em comum.

CAPÍTULO X

CORREÇÃO E OUTRAS QUALIDADES DE UM ALGORITMO

A correção é uma qualidade diferente das demais que podem ser consideradas com relação a um algoritmo. Enquanto as outras qualidades são relativas, isto é, podem ser mais importantes em alguns algoritmos e menos em outros, a correção é condição para que o algoritmo tenha sentido de existência.

As outras qualidades podem interessar em um grau maior em algum algoritmo e em um grau menor em outros, enquanto a correção é uma qualidade absoluta. Só interessa um algoritmo 100% correto.

As fontes mais freqüentes de erros em algoritmos são as falhas humanas, que se manifestam não somente na lógica incorreta ou na ausência de elementos importantes, mas também na simples transcrição errônea e em outros erros ou esquecimentos banais.

Na construção de um algoritmo há, pois, uma hierarquia de considerações a fazer:

1. CORREÇÃO

O algoritmo deve resolver correta e completamente o problema do usuário.

Como os mecanismos para se provar a correção de um algoritmo ainda não estão disponíveis ou seu uso é economicamente inviável e não inspiram confiança, a questão passa a ser, como garantir a construção de algoritmos que sejam corretos ?

A tarefa de provar a correção de um algoritmo, assim como de um programa, ainda é uma tarefa complicada, trabalhosa e, em muitos casos, impossível de ser realizada.

Assim, para garantir que todas as providências razoáveis e disponíveis tenham sido tomadas deve-se, pelo menos, adotar as seguintes práticas:

- Utilizar a técnica da modularização dos problemas de sorte a obter subproblemas menores, menos complexos, com menor número de variáveis e com poucos caminhos de controle (caminhos do início ao fim do algoritmo). Como um problema menor, menos complexo é mais facilmente bem compreendido, é muito mais provável que se consiga escrever uma solução que atenda os requisitos pretendidos;
- Fazer-se uma verificação intuitiva da correção, antes de cada novo particionamento do problema em subproblemas, isto é, deve-se parar, analisar, verificar consistência, completude e adequação do que está sendo produzido. Se necessário obter a opinião e colaboração de terceiros;
- Fazer uma análise para descobrir quais os dados mais significativos ao contexto do problema, os mais utilizados por quem vai usar a solução. Aplicar testes suficientes para garantir, pelo menos, que o algoritmo produz os resultados esperados para os dados que representam o uso mais comum e freqüente.

2. EFETIVIDADE ou EFICÁCIA

Diz respeito à capacidade de o algoritmo chegar a um resultado em um tempo hábil.

3. EFICIÊNCIA E CLAREZA

Deve fazê-lo da forma mais econômica possível tanto do ponto de vista da máquina quanto do ponto de vista das pessoas que vão usá-lo (possivelmente com adaptações). O algoritmo mais eficiente é aquele que obtém o resultado desejado com a melhor relação custo/benefício.

Clareza se refere à capacidade de o algoritmo expressar as ações que devem ser realizadas sem que haja ambigüidades.

4. GENERALIDADE E ADAPTABILIDADE

Servir como solução para uma classe de problemas e não somente para um problema específico. Prever, tanto quanto possível a necessidade de futuras adaptações estando preparado para aceitá-las sem grandes dificuldades.

FORMAS DE MELHORIA DA EFICIÊNCIA DE UM ALGORITMO

A primeira preocupação com a construção de qualquer algoritmo é construí-los corretos, pois, sem esta qualidade as outras perdem a sua razão de ser. Depois de se ter um algoritmo correto pode-se pensar em melhorar-lhe a eficiência. Algumas cuidados na melhoria da eficiência de um algoritmo são, a seguir, apresentadas:

a) Efetuar os cálculos que não são afetados pela repetição, fora da repetição.

Exemplo:

```
Algoritmo Exemplo
  m, n, i : inteiro
  x, y, z, k : real
Início
   Ler ( m, n )
   Para i = 1 até 5
       Início
           x = m/2
           y = n x 2
           z = (m + n) / 2
           Para k = x até y  inc z
               Escrever ( k/i )
           Fim_Para k
       Fim_Para i
Fim
```

Como os cálculos de x, y e z que interessam para a variação de k não dependem do valor de i, podem ser feitos uma única vez antes de entrar na primeira repetição e não 5 vezes como acontece no exemplo acima.

Assim temos a nova versão melhorada com relação à eficiência como segue abaixo:

```
Algoritmo Exemplo
  m,n,i : inteiro
  x, y, z, k : real
Início
   Ler ( m, n )
   x = m/2
   y = n x 2
   z = (m + n) / 2
   Para i = 1 até 5
       Para k = x até y  inc z
           Escrever ( k/i )
       Fim_Para k
   Fim_Para i
Fim
```

b) Evitar de calcular a mesma expressão várias vezes sem necessidade.

Exemplo:

```
Algoritmo Equação Grau 2
  a, b, c, x1, x2 : real
Início
   Ler ( a, b, c )
   Enquanto a ≠ 0
       Início
           Se b² - 4ac ≥ 0
           então Início
                   x1 = (-b + √(b² - 4ac)) / 2a

                   x2 = (-b - √(b² - 4ac)) / 2a
                   Escrever ( x1, x2 )
               Fim
           Ler ( a, b, c )
       Fim_enquanto
Fim
```

A expressão $\mathbf{b^2 - 4ac}$ neste caso é calculada 3 vezes quando poderíamos calculá-la uma única vez como é mostrado na versão a seguir:

```
Algoritmo Equação Grau 2
  a, b, c, d, x1, x2 : real
Início
   Ler ( a, b, c )
   Enquanto a ≠ 0
       Início
           d = b² – 4ac
           Se d ≥ 0 então Início
                   x1 = (-b + √d) / 2a
                   x2 = (-b - √d) / 2a
                   Escrever ( x1, x2 )
               Fim
           Ler ( a, b, c )
       Fim_enquanto
Fim
```

Obs.: Eficiência se refere tanto à maior velocidade do processamento quanto ao menor uso de espaço de memória. Geralmente estas duas eficiências são conflitantes. Conseguimos maior velocidade de processamento às custas de um uso de mais memória e quando tentamos economizar memória, geralmente, aumentamos o tempo de processamento.

c) Não esgotar um "Loop" quando não for necessário.

Exemplo:

```
Algoritmo Verifica Primo
  m, i, c : inteiro
Início
    Ler ( m )
    Enquanto m > 0
        Início
            c = 2
            Para i = 2 até √m
                Se m mod i = 0 então c = c + 1
            Fim_Para i
            Se c = 2 então escrever ( m , "E PRIMO" )
                      senão escrever ( m, "NÃO E PRIMO" )
            Ler ( m )
        Fim_enquanto
Fim
```

Como neste problema se deseja apenas saber se o m lido é Primo ou não, não há necessidade de esgotar-se todo o "Loop". Ao primeiro divisor encontrado no intervalo de 2 até a raiz quadrada de m já se sabe que m não é primo.

Versão melhorada do algoritmo Verifica Primo

```
Algoritmo Verifica Primo
  m, i, c : inteiro
Início
    Ler ( m )
    Enquanto m > 0
        Início
            c = 2
            i = 2
            Enquanto c = 2
                Início
                    Se m mod i = 0 então c = c + 1
                    i = i + 1
                Fim_enquanto
            Se c = 2 então escrever ( m, "E PRIMO" )
                      senão escrever ( m, "NÃO E PRIMO" )
            Ler  (m )
        Fim_enquanto
Fim
```

d) Não examinar condições desnecessárias.

Exemplo:

```
Algoritmo Conta Intervalos
   m, c1, c2, c3, c4 : inteiro
Início
    Ler  ( m )
    Enquanto m > 0
        Início
            Se m > 0 e m ≤ 10 então c1 = c1 + 1
            Se m > 10 e m ≤ 25 então c2 = c2 + 1
            Se m > 25 e m ≤ 50 então c3 = c3 + 1
            Se m > 50 e m ≤ 75 então c4 = c4 + 1
            Ler m
        Fim_enquanto
    Escrever ( c1, c2, c3, c4 )
Fim
```

Solução melhorada do algoritmo Conta Intervalos.

```
Algoritmo Conta Intervalo
  m, c1, c2, c3, c4 : inteiro
Início
    c1=c2=c3=c4=0
   Ler ( m )
   Enquanto m > 0
       Início
           Se m > 50 e m ≤ 75
           então c4 = c4 + 1
           senão Se m > 25
                 então c3 = c3 + 1
                 senão Se m > 10
                       então c2 = c2 + 1
                       senão c1 = c1 + 1
           Ler m
       Fim_enquanto
   Escrever ( c1, c2, c3, c4 )
Fim
```

Os algoritmos acima ambos fazem a mesma coisa, isto é, lêem um número qualquer de valores m e verificam quantos deles estão em cada um dos seguintes intervalos (0, 10], (10, 25], (25, 50] e (50, 75]. Na primeira versão deste algoritmo são examinadas 4 condições duplas para cada valor de m lido. Na versão melhorada, se m estiver no quarto intervalo é examinada apenas uma condição dupla. No pior caso são examinadas uma condição dupla e duas simples. Se a quantidade de valores m lidos for pequena, isto naturalmente não é muito significativo, mas para um número muito grande de valores examinados isto vai fazer diferença.

e) Escolher métodos mais eficientes quando existem várias opções.

Tomando como exemplo a ordenação de um conjunto em ordem crescente. Se o conjunto que tivermos for pequeno, não vale à pena preocupar-se com o uso de um algoritmo mais eficiente, pois, para conjuntos pequenos todos eles terão desempenhos semelhantes. Pode até acontecer que o algoritmo mais eficiente na teoria, para um determinado caso prático, seja menos eficiente que outro que teoricamente é pior. Contudo, em conjuntos grandes a diferença pode também ser bem grande.

APÊNDICE A

TRADUÇÃO DE ALGORITMOS PARA PASCAL

I – Constantes em PASCAL

As constantes em PASCAL podem ser de três tipos básicos: Numéricas, Lógicas e Literais.

Numéricas:

a) inteiras : **120**, **+561**, **-240**

b) reais:

b.1) Notação Comum : **12.35**, **-.5**, **+5.**

b.2) Notação Científica : **1.2E2**, **-0.56E-5**, **+1.78E+11**

Lógicas :

As constantes lógicas são duas : **true** e **false**.

Literais:

É qualquer conjunto de caracteres colocado entre apóstofos.

Ex.: **'João da Silva'**, **'X100(-.#Ab'**, **'120'**, **''''**

O último representa o caractere apóstrofo quando se quer usá-lo como parte da constante literal.

II – Variáveis em PASCAL

Variável é o nome dado a uma localização de memória na qual é possível armazenar-se o valor de uma constante.

Em PASCAL as regras de formação dos nomes de variáveis são as seguintes:

1. O primeiro caractere do nome deve ser uma letra de A até Z, maiúscula ou minúscula, pois, a maioria dos Compiladores PASCAL não fazem distinção entre letra maiúscula e minúscula;
2. Os demais caracteres, se houver, devem ser letras ou dígitos ou o símbolo de quebra "_".
3. O número de caracteres do nome da variável não é limitado. Contudo, em algumas implementações o Compilador só reconhece até o caractere 63. Os demais caracteres são ignorados.

Portanto, são nomes de variáveis válidos em PASCAL :

A

x

a1

valor_de_variavel

d12

Não são válidos, por exemplo:

Media aritmética	(não são aceitos espaços em branco no nome de uma variável)
n°	(apenas letras ou dígitos são aceitos)
5a	(o primeiro caractere do nome deve ser letra)
A1.5	(ponto não é nem letra e nem dígito)

III Expressões em PASCAL

Antes de mostrar a tradução de algumas expressões aritméticas para PASCAL é importante chamar a atenção para alguns pontos:

1. Quando existir numerador ou denominador de fração que não seja simples, isto é, que envolve uma ou mais operações, na tradução para PASCAL este numerador ou denominador deve ser colocado entre parênteses;
2. O PASCAL não usa colchetes ou chaves em expressões aritméticas como elementos de agrupamento de operações, portanto, estes símbolos se aparecerem devem ser substituídos por parênteses, podendo-se ter vários níveis de parênteses;
3. Na matemática, em muitas ocasiões, os operadores podem ficar implícitos. Em PASCAL nenhum operador pode ficar implícito.
4. Os operadores aritméticos em PASCAL são os apresentados abaixo:

   4.1. Adição → +

   4.2. Subtração → -

   4.3. Multiplicação → *

   4.4. Divisão → /

   4.5. Quociente inteiro da divisão → div

   4.6. Resto inteiro da divisão → mod

5. O operador de atribuição em PASCAL é o " := "
6. Operações para as quais não há operadores são resolvidas por meio de funções, das quais as mais usuais já estão pré-definidas. No caso do PASCAL, as principais funções pré-definidas são :

SQR (X)	→ quadrado de X	tipo real ou inteiro
LN (X)	→ logaritmo natural de X	tipo real
EXP (X)	→ exponencial de X	tipo real
ABS (X)	→ valor absoluto de X	tipo real ou inteiro
SQRT(X)	→ raiz quadrada de X	tipo real
TRUNC(X)	→ trunca o valor de X	tipo inteiro
ROUND(X)	→ arredonda o valor de X	tipo inteiro
SIN (X)	→ seno de X	tipo real
COS (X)	→ cosseno de X	tipo real
TAN (X)	→ tangente de X	tipo real

7. Os operadores relacionais em PASCAL são os seguintes:

   = → =

   < → <

   > → >

   ≥ → >=

   ≤ → <=

   ≠ → <>

8. Os operadores lógicos em PASCAL são os abaixo listados:

   não → not

   e → and

   ou → or

9. A prioridade dos operadores em PASCAL é a seguinte:

   1) Funções
   2) *, /, div, mod, and
   3) +, -, or
   4) =, <>, <=, <, >, >=
   5) :=
   6) Parênteses podem alterar esta ordem.
   7) Da esquerda para a direita em caso de indeterminação.

Exemplos de expressões aritméticas e sua tradução para PASCAL:

$$x = \frac{a^2bc}{2} + \frac{a}{b^2+5}$$

X:= (SQR(A)*B*C)/2 + A/ (SQR(B) + 5);

$$m = (a+b^2)[(x+y^3)^2 - w] + d - 1$$

M:= (A + SQR (B)) * (SQR (X + SQR (Y) * Y) – W) + D – 1;

$$d = \sqrt{\sqrt{|a-b|}} + 1$$

D := SQRT (SQRT (ABS (A – B))) + 1;

x = $sen^2x + \cos x^2 - 1$

X := SQR (SIN (X)) + COS (SQR (X)) – 1;

$$y = \frac{a}{b} + \frac{m-n}{n+1}$$

Y := A/B + (M-N)/ (N + 1);

IV – Linguagem algoritmica x Linguagem PASCAL

Definição de variáveis:

m, n, i : inteiro	→ **var** m, n, i : **integer**;
a, b, c : real	→ **var** a, b, c : **real**;
v1, v2 : lógico	→ **var** v1, v2 : **boolean**;
nome, ender : literal	→ **var** nome, ender : **char**;

V(10): vetor de inteiro → **Type** Vetor = **array**[1..10] **of integer**;
V: Vetor;

M(5,5) : matriz de real → **Type** Matriz = **array**[1..5,1..5] **of real**;
M : Matriz;

Empregado Registro
Nome(30): Literal
Idade : Inteiro
Salário: Real
Fim_Registro

→ **Type** Registro = **Record**
Nome: **string**[30];
Idade: **Integer**;
Salario: **Real**;
end;
Var Empregado : Registro;

Definição de comandos:

Fim → **End.** (final de Programa)
Ou
End; (final de bloco)

Ler (a, b, c) → **Read** (a, b, c);

Escrever (x, y, (x+y)/3, "Resultado") → **Write** (x, y, (x+y)/3, 'Resultado');

x = 5 → x := 5;

$d = b^2-4ac$ → d := **Sqr** (b) – 4 * a * c ;

Se $x \neq 0$ e y = 1 então ler (a, b, c) → **If** (x <> 0) **and** (y = 1) **then Read** (a, b, c);

Se $b^2-4ac < 0$ ou a = 0 então x = a + b
senão x = a – b

→ **If** (**Sqr** (b)-4*a*c < 0) **or** (a = 0) **then** x:= a + b
else x:= a – b ;

Se i mod 2 = 0 então Início
ma = a
Se ma < b então ma = b
Se ma < c então ma = c
Escrever (ma, "E o maior")
Fim

→ **If** i mod 2 = 0 **then Begin**
ma := a;
if ma < b **then** ma:= b;
if ma < c **then** ma:= c;
Write (ma, 'E o maior');
End;

```
i = 0
Enquanto i < 5
    Início
        Ler ( a, b, c )
        me = a
         Se me > b então me = b
         Se me > c então me = c
         Escrever ( me , " É o menor ")
         i := i + 1
   Fim_enquanto
```

→

```
i:= 0;
While i < 5 do
    begin
        Read ( a, b, c );
        me := a;
        if me > b then me:= b;
        if me > c then me:= c;
        write ( me, ' E o menor' );
        i := i + 1;
    end;
```

```
Para i = 1 até 10
     Ler ( V( i ) )
 Fim_Para i
```

→

```
For i := 1 to 10 do
        Read ( V[ i ] );
```

```
Para val = 1 até m
    Início
        Soma = Soma + m
        m = m + 1
    Fim_Para val
```

→

```
For val := 1 to m do

    Begin
        Soma := Soma + m ;
        m := m + 1;
    End;
```

```
Para n = 10 até 1
     Escrever ( n x n )
Fim_Para n
```

→

```
For n := 10 downto 1 do
    write ( n*n );
```

```
Algoritmo Fatorial
c,m, i, fat : Inteiro
Início
    Para c = 1 até 10
        Início
          Ler  ( m )
          fat = 1
          Para i = 1 até m
              fat = fat x i
          Fim_Para i
          Escrever ( m, fat )
       Fim_Para c
Fim
```

→

```
Program Fatorial;
   Var  c, m, i, fat : integer;
Begin
    For c := 1 to 10 do
        Begin
            Read ( m );
            fat := 1;
            For i := 1 to m do
                fat := fat * i;
            write ( m , fat );
        End;
End.
```

```
Algoritmo Raizes
a, b, c, x, x1, x2, d : Reais
Início
    Ler ( a, b, c )
    Se a = 0 então se b ≠ 0      então      Início
                                                x = -c/b
                                                Escrever ( x )
                                            Fim
                                 senão      Escrever ( c )
            senão Início
                    d = b^2 - 4ac
                    Se d < 0 então Escrever  ( " Equação sem raízes reais " )
                            senão     Início
                                          x1 = (-b + √d) / 2a
                                          x2 = (-b - √d) / 2a
                                          Escrever ( x1, x2 )

                                      Fim
                    Fim
Fim
```

```
Program Raizes;
   Var a, b, c, x, x1, x2, d : real;
Begin
    Read ( a, b, c );
    If ( a = 0 ) then if b <> 0 then begin
                                        x := -c/b;
                                        write ( x );
                                      end
                                 else write (c )
                else begin
                        d := b*b-4*a*c;
                        If d < 0 then write ( ' Equação sem raizes reais')
                                 else begin
                                        x1 := (-b + Sqrt( d ))/(2 * a );
                                        x2 := (-b - Sqrt( d ))/(2 * a );
                                        write ( x1, x2 );
                                      end;
End.
```

```
Algoritmo OrdenaCrescenteBolha
V ( 6 ) : Mat de tipo Inteiro
i, k, x : Inteiro
Início
    Para i = 1 até 6
        Início
            Ler (V( i ) )
            Escrever ( V (i ) )
        Fim_Para i
    k = 0
    Enquanto k = 0
        Início
            k = 1
            Para i = 1 até 5
            Se V(i) > V(i+1) então Início
                                    x = V(i)
                                    V(i) =  V(i+1)
                                    V(i+1) = x
                                    k = 0
                                  Fim
            Fim_Para i
        Fim_enquanto
    Para i = 1 até 6
        Escrever ( V( i ) )
    Fim_Para i
Fim
```

→

```
Program OrdenaCrescenteBolha;
Var V: array[1..6] of integer;
      i, k, x : integer;
Begin
    For i := 1 to 6 do
    Begin
        Read ( V[ i ] );
         Write (  V [ i ] );
    End;
    k := 0;
    While k = 0 do
     Begin
        k := 1;
         For i := 1 to 5 do
            If V [ i ] > V[ i+1 ]
            Then Begin
                    x := V [ i ] ;
                    V[ i ] := V[ i + 1 ];
                    V [ i + 1 ] := x;
                     k := 0;
                 End;
     End;
     For i := 1 to 6 do
        Write ( V[ i ], ‘   ’ );
End.
```

```
Algoritmo Cálculo-do-Fatorial
  m, i, fat : inteiro
Início
    i = 1
    Enquanto i ≤ 30
        Início
            Ler ( m )
            Fat = Fatorial ( m )
            Escrever ( m, fat )
            i = i + 1
        Fim_enquanto
Fim

Função Fatorial ( n: inteiro ): inteiro
  f, i : inteiro
Início
    i = 1
    Enquanto i ≤ n
        Início
            f = f x i
            i = i + 1
        Fim_enquanto
    Fatorial = f
    Retorne
Fim
```

→

```
Program Calculo_do_Fatorial ;
  Var m, i, fat : integer;
   Function Fatorial ( n: integer ): integer;
    Var f, i : integer;
    Begin
        i := 1;
        While i <= n do
         Begin
            f := f * i;
            i := i + 1;
         End;
        Fatorial := f ;
    End;
 Begin
    i := 1;
    While i <= 30 do
    Begin
        Read ( m );
        fat := Fatorial ( m );
        writeln ( m, ‘    ’, fat );
        i := i + 1;
    End;
 End.
```

```
Função MaiorVet ( Vet: vetor, n : inteiro ): inteiro
  maior, i : inteiro
Início
    maior = Vet ( 1 )
     Para i = 2 até n
         Se Vet( i ) > maior então maior = Vet ( i )
     Fim_Para i
     MaiorVet = maior
     Retorne
Fim
```

↓

```
Function  Maior_Vet (V: vetor, n : integer ): integer;
  Var  maior, i : integer;
Begin
     maior := V[1];
     for  i := 2 to  n do
        if  maior < V[ i ] then  maior := V[ i ];
     Maior_Vet : = maior;
End;
```

APÊNDICE B

TRADUÇÃO DE ALGORITMOS PARA C

I – Constantes em C

As constantes em C podem ser de três tipos básicos: Numéricas, Lógicas e Literais.

Numéricas:

c) inteiras : **120**, **+561**, **-240**

d) reais:

b.1) Notação Comum : **12.35**, **-.5**, **+5.**

b.2) Notação Científica : **1.2E2**, **-0.56E-5**, **+1.78E+11**

Em C não existem constantes lógicas. Tudo o que não é zero é **true**. Só o que for zero é **false**.

Literais:

É qualquer conjunto de caracteres colocado entre aspas.

Ex.: "**João da Silva"**, "**X100(-.#Ab"**, "**120"**, **""**"

O último representa o caractere aspas quando se quer usá-lo como parte da constante literal.

II – Variáveis em C

Variável é o nome dado a uma localização de memória na qual é possível armazenar-se o valor de uma constante.

Em C as regras de formação dos nomes de variáveis são as seguintes:

4. O primeiro caractere do nome deve ser uma letra de A até Z, maiúscula ou minúscula. O Compilador C faz distinção entre letra maiúscula e minúscula. Isto exige algum cuidado para não se usar variáveis diferentes com mesmo objetivo;
5. Os demais caracteres, se houver, devem ser letras ou dígitos ou o símbolo de quebra "_".
6. O número de caracteres do nome da variável não é limitado. Contudo, em algumas implementações o Compilador só reconhece até o caractere 63. Os demais caracteres são ignorados.

Portanto, são nomes de variáveis válidos em C :

A

a

a1

valor_de_variavel

Valor_de_variavel

Valor_de_Variável

Tanto o **A** e o **a** são variáveis diferentes, quanto o são as três últimas apresentadas.

Não são válidos, por exemplo:

Media aritmética	(não são aceitos espaços em branco no nome de uma variável)
n°	(apenas letras ou dígitos são aceitos)
5a	(o primeiro caractere do nome deve ser letra)
A1.5	(ponto não é nem letra e nem dígito)

III Expressões em C

Antes de mostrar a tradução de algumas expressões aritméticas para C é importante chamar a atenção para alguns pontos:

1. Quando existir numerador ou denominador de fração que não seja simples, isto é, que envolve uma ou mais operações, na tradução para C este numerador ou denominador deve ser colocado entre parênteses;
2. O C não usa colchetes ou chaves em expressões aritméticas como elementos de agrupamento de operações, portanto, estes símbolos se aparecerem devem ser substituídos por parênteses, podendo-se ter vários níveis de parênteses;
3. Na matemática, em muitas ocasiões, os operadores podem ficar implícitos. Em C nenhum operador pode ficar implícito.
4. Os operadores aritméticos em C são os apresentados abaixo:

Adição	→ +
Subtração	→ -
Multiplicação	→ *
Divisão	→ /
Resto inteiro da divisão	→ %
Incremento	→ ++
Decremento	→ --

5. O operador de atribuição em C é o " = "
6. Operações para as quais não há operadores são resolvidas por meio de funções, das quais as mais usuais já estão pré-definidas. No caso do C, as principais funções pré-definidas são :

pow (base, expoente)	→ potenciação	tipo real ou inteiro
log (X)	→ logaritmo natural de X	tipo real
exp (X)	→ exponencial de X	tipo real
fabs (X)	→ valor absoluto de X	tipo real ou inteiro
sqrt(X)	→ raiz quadrada de X	tipo real
ceil (X)	→ menor inteiro contido em X	tipo inteiro
fmod(X, Y)	→ resto inteiro de X dividido por y	tipo inteiro
sin (X)	→ seno de X	tipo real
cos (X)	→ cosseno de X	tipo real
tan (X)	→ tangente de X	tipo real
asin(X)	→ arco seno de x	tipo real
atan (X)	→ arco tangente de X	tipo real

7. Os operadores relacionais em C são os seguintes:

= → ==

< → <

> → >

≥ → >=

≤ → <=

≠ → !=

8. Os operadores lógicos em C são os abaixo listados:

não → !

e → &&

ou → ||

9. A prioridade dos operadores em C é a seguinte:

9.1 Funções

9.2 ++, --

9.3 *, /, %, &&

9.4 +, -, ||

9.5 ==, !=, <=, <, >, >=

9.6 Parênteses podem alterar esta ordem.

9.7 Da esquerda para a direita em caso de indeterminação.

Exemplos de expressões aritméticas e sua tradução para C:

$$x = \frac{a^2bc}{2} + \frac{a}{b^2+5}$$

x= (pow(a, 2) * b * c) / 2 + a/ (pow (b, 2) + 5);

$$m = (a+b^2)[(x+y^3)^2 - w] + d - 1$$

m = (a + pow (b, 2) * (pow (x + pow (y, 3), 2) – w) + d – 1;

$$d = \sqrt{\sqrt{|a-b|}} + 1$$

d = sqrt (sqrt (fabs (a – b))) + 1;

x = sen^2x + cos x^2 –1

x = pow (sin (x), 2) + cos (pow (x, 2)) – 1;

$$y = \frac{a}{b} + \frac{m-n}{n+1}$$

y = a / b + (m – n) / (n + 1);

IV – Linguagem algorítmica x Linguagem C

Definição de variáveis:

m, n, i : inteiro	→ **int** m, n, i ;
a, b, c : real	→ **float** a, b, c ;
nome, ender : literal	→ **char** nome, ender ;
V(10): vetor de inteiro	→ **int** V[10];
M(5,5) : matriz de real	→ **float** M[5][5];
Empregado Registro	→ **struct** Empregado
	{
Nome(30): Literal	**char** Nome[30];
Idade : Inteiro	**int** Idade;
Salário: Real	**float** Salario;
Fim_Registro	**}**

Definição de comandos:

Início	→ **{**
Fim	→ **}**
Ler (a, b, c)	→ **scanf** (“%d%d%d”, &a, &b, &c);
Escrever (x, y, (x+y)/3, “Resultado”)	→ **printf** (“%d %d %d Resultado”,x, y, (x+y)/3);
x = 5	→ x = 5;
d = b^2-4ac	→ d = **pow** (b ,2) – 4 * a * c ;
Se x ≠ 0 e y = 1 então ler (a, b, c)	→ **if** ((x != 0) && (y == 1))**scanf** (“%f%f%f”,&a, &b, &c);
Se b^2-4ac < 0 ou a = 0 então x = a + b	→ **if** (**pow** (b, 2) – 4 * a * c < 0) || (a == 0) x = a + b;
senão x = a – b	**else** x = a – b ;

```
Se i mod 2 = 0 então Início
                    ma = a
                    Se ma < b então ma = b
                    Se ma < c então ma = c
                    Escrever (ma, "E o maior")
                  Fim
```

→

```
if (i % 2 == 0) {
    ma = a;
    if (ma < b) ma= b;
    if (ma < c ) ma= c;
    printf ("%d E o maior", ma );
}
```

```
Caso  i   for
Início
    1: x = a + b

    2: x = a – b

    3: x = a x b

    4: x = a / b

    senão x = 0
Fim
```

→

```
switch ( i )
    {
        case 1: x = a + b;
                break;
        case 2: x = a – b;
                break;
        case 3: x = a * b;
                break;
        case 4: x = a / b;
                break;
        default  x = 0;
    }
```

```
i = 0
Enquanto i < 5
    Início
        Ler (a, b, c )
        me = a
        Se me > b então me = b
        Se me > c então me = c
        Escrever (me , " É o menor ")
        i := i + 1
    Fim_enquanto
```

→

```
i = 0;
while ( i < 5 )
    {
        scanf ( "%d%d%d", &a,&b, &c );
        me = a;
        if (me > b ) me= b;
        if (me > c ) me= c;
        printf ("%d   E o menor ", me );
        i = i + 1;
    }
```

```
Para i = 1 até 10
    Ler ( V( i ) )
Fim_Para i
```

→

```
for( i = 1; i <= 10; i++ )
    scanf ("%d", &V[ i ] );
```

```
Para val = 1 até m
    Início
        Soma = Soma + m
        m = m + 1
    Fim_Para val
```

→

```
for (val = 1; val <= m; val++)
    {
        Soma = Soma + m;
        m = m = 1;
    }
```

```
Para n = 10 até 1
    Escrever ( n x n )
Fim_Para n
```

→

```
For ( n = 10; n >= 1; n--)
    printf ("%d", n*n );
```

Observação

Para se escrever um programa em C costumam-se utilizar uma quantidade de bibliotecas do C já prontas, das quais serão apresentadas somente as mais usuais:

```
# include <stdio.h>
# include <conio.h>
# include <math.h>
```

Um programa em C é sempre uma função. No caso do Programa Principal, uma função que não retorna nada, isto é **void.**

```
Algoritmo Fatorial
c,m, i, fat : Inteiro
Início
    Para c = 1 até 10
        Início
            Ler ( m )
            fat = 1
            Para i = 1 até m
                fat = fat x i
            Fim_Para i
            Escrever ( m, fat )
        Fim_Para c
Fim
```

→

```
#include <stdio.h>
#include <conio.h>

void main( );
{
   int  c, m, i, fat ;
   for ( c = 1; c <= 10; c++)
        {
            scanf ("%d", &m );
            fat = 1;
            for (i = 1; i <= m; i++)
                fat = fat * i;
            printf ("%d   %d ", m , fat );
        }
}
```

```
Algoritmo Raizes
a, b, c, x, x1, x2, d : Reais
Início
    Ler ( a, b, c )
    Se a = 0 então se b ≠ 0      então      Início
                                                x = -c/b
                                                Escrever ( x )
                                            Fim
                                 senão      Escrever ( c )
            senão Início
                    d = b^2 - 4ac
                    Se d < 0 então Escrever ( " Equação sem raizes reais" )
                            senão     Início
                                          x1 = (-b + √d) / 2a
                                          x2 = (-b - √d) / 2a
                                          Escrever ( x1, x2 )

                                      Fim
                    Fim
Fim
```

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

void main ( )
 {
   float a, b, c, x, x1, x2, d ;
   scanf ("%f%f%f",  &a, &b, &c );
   if ( a == 0 )  if  ( b != 0) {
                                    x = -c/b;
                                    printf ("%f ", x );
                                }
                  else printf (" %f ", c );
   else {
            d = b*b-4*a*c;
            If ( d < 0 ) printf ( " Equação sem raizes reais   ");
             else {
                    x1 = ( -b + Sqrt ( d ) ) / ( 2 * a );
                    x2 = ( -b – Sqrt ( d ) ) / ( 2 * a );
                    printf (" %f  %f ", x1, x2 );
                  }
}
```

```
Algoritmo OrdenaCrescenteBolha
V ( 6 ) : Mat de tipo Inteiro
i, k, x : Inteiro
Início
    Para i = 1 até 6
        Início
            Ler  (V( i ) )
            Escrever (V (i ) )
        Fim_Para i
    k = 0
    Enquanto k = 0
       Início
         k = 1
         Para i = 1 até 5
            Se V(i) > V(i+1) então Início
                                    x = V(i)
                                    V(i) =  V(i+1)
                                    V(i+1) = x
                                    k = 0
                                  Fim
        Fim_Para i
     Fim_enquanto
    Para i = 1 até 6
        Escrever (V( i ) )
    Fim_Para i
Fim
```

→

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
    int  V{6], i, k, x ;
    for ( i = 1; i <= 6; i++)
    {
        scanf (“%d“, V[ i ] );
        printf(“%d   “, V[ i ] );
    }
     k = 0;
     while ( k = 0 )
     {
          k = 1;
          for ( i = 1; i <= 5; i++ )
              if ( V[ i ] > V[ i+1 ] )
              {
                      x= V[ i ];
                      V[ i ] = V[ i+1 ];
                      V[ i+1 ] = x;
                       k = 0;
              }
      }
      for ( i = 1; i <= 6; i++ )
          printf (“%d  ”; V[ i ] );
}
```

```
Algoritmo Cálculo-do-Fatorial
   m, i, fat : inteiro
Início
    i = 1
    Enquanto i ≤ 30
        Início
            Ler ( m )
            Fat = Fatorial ( m )
            Escrever ( m, fat )
            i = i + 1
        Fim_enquanto
Fim

Função Fatorial ( n: inteiro ): inteiro
   f, i : inteiro
Início
     i = 1
     Enquanto i ≤ n
         Início
             f = f x i
             i = i + 1
         Fim_enquanto
     Fatorial = f
     Retorne
Fim
```

→

```
#include <stdio.h>
void main ( )
{
  int  m, i, fat ;
   int Fatorial ( int n )
   {
     int  f, i ;
     i = 1;
     while ( i <= n )
        {
            f = f * i;
            i = i + 1;
         }
      return ( f ) ;
    }
   i = 1;
   while ( i <= 30)
    {
         scanf (“%d”, &m );
         fat = Fatorial ( m );
         printf (“\n %d   %d ” , m,  fat );
         i = i + 1;
     }
 }
```

```
Função MaiorVet ( Vet: vetor, n : inteiro ): inteiro
  maior, i : inteiro
Início
    maior = Vet ( 1 )
     Para i = 2 até n
         Se Vet( i ) > maior então maior = Vet ( i )
     Fim_Para i
     MaiorVet = maior
     Retorne
Fim
```

↓

```
Int Maior_Vet ( V, int n )
{
     int maior, i ;
     maior = V[1];
     for ( i = 2; i ≤ n; i++)
        if ( maior < V[ i ] ) maior = V[ i ];
     Return ( maior );
}
```

BIBLIOGRAFIA UTILIZADA

1. ORTH, A . I. Algoritmos. Editora Pallotti. Porto Alegre, 1987.
2. FARRER, H. et alii . Algoritmos Estruturados. Editora Livros Técnicos e Científicos. Rio de Janeiro, 3ª edição, 1999.
3. VENÂNCIO, C. F. Desenvolvimento de Algoritmos: Uma nova abordagem. Editora Érica. São Paulo, 1997.
4. SALVETTI, D. D. & BARBOSA, L. M. Algoritmos. Editora Makron Books. São Paulo, 1998.
5. PINTO, W. S. Introdução ao Desenvolvimento de Algoritmos e Estruturas de Dados. Editoa Érica. São Paulo, 10ª Edição, 1999.
6. VELOSO, P. et alii Estruturas de Dados. 14ª edição. Editora Campus. Rio de Janeiro, 1983.
7. MIZRAHI, V. V. Treinamento em Linguagem C: Curso Completo. Vols 1 e 2. Editora Makron Books. São Paulo, 1990.
8. ZIVIANI, N. Projeto de Algoritmos com Implementações em Pascal e C. Editora Pioneira. São Paulo, 1996.
9. TREMBLAY, J. P. BUNT, R. B. Ciência dos Computadores: Uma abordagem algorítmica. Editora McGraw-Hill. São Paulo, 1978.
10. DIJKSTRA, E. W. A Discipline of Programming. Englewood Gliffs. New Jersey. Prentice-Hall, 1976.
11. WIRTH, N. From Programming Techniques to Programming Methods. North-Holland, 1974.
12. KNUTH, D. E. The Art of Progamming vol 1 Fundamental Algorithms. Addison-Wesley. Massachussets, 1968.
13. TERADA, R. Desenvolvimento de Algoritmos e Estruturas de Dados. Editora McGraw-Hill. São Paulo, 1991.
14. SCHILDT,H. C Completo e Total. Editora Makron Books, São Paulo, 1997.
15. FORBELLONE, A .L.V. & EBERSPACHER, H.F. Lógica de Programação. Editora Makron Books, São Paulo, 2000.